The 30 Year Horizon

Manuel Bronstein
James Davenport
Albrecht Fortenbacher
Jocelyn Guidry
Michael Monagan
Jonathan Steinbach
Stephen Watt

William Burge
Michael Dewar
Patrizia Gianni
Richard Jenks
Scott Morrison
Robert Sutor
Jim Wen

Timothy Daly
Martin Dunstan
Johannes Grabmeier
Larry Lambe
William Sit
Barry Trager
Clifton Williamson

Volume 5: Axiom Interpreter
Portions Copyright (c) 2005 Timothy Daly

The Blue Bayou image Copyright (c) 2004 Jocelyn Guidry

Portions Copyright (c) 2004 Martin Dunstan
Portions Copyright (c) 2007 Alfredo Portes
Portions Copyright (c) 2007 Arthur Ralfs
Portions Copyright (c) 2005 Timothy Daly

Portions Copyright (c) 1991-2002, The Numerical ALgorithms Group Ltd.
All rights reserved.

This book and the Axiom software is licensed as follows:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of The Numerical ALgorithms Group Ltd. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Inclusion of names in the list of credits is based on historical information and is as accurate as possible. Inclusion of names does not in any way imply an endorsement but represents historical influence on Axiom development.

Michael Albaugh  Cyril Alberga  Roy Adler
Christian Aistleitner  Richard Anderson  George Andrews
S.J. Atkins  Henry Baker  Martin Baker
Stephen Balzac  Yuri Baransky  David R. Barton
Thomas Baruchel  Gerald Baumgartner  Gilbert Baumslag
Michael Becker  Nelson H. F. Beebe  Jay Belanger
David Bindel  Fred Blair  Vladimir Bondarenko
Mark Botch  Raoul Bourquin  Alexandre Bouyer
Karen Braman  Peter A. Broadbery  Martin Brock
Manuel Bronstein  Stephen Buchwald  Florian Bundschuh
Luanne Burns  William Burge  Ralph Byers
Quentin Carpent  Robert Caviness  Bruce Char
Ondrej Certik  Tzu-Yi Chen  Cheekai Chin
David V. Chudnovsky  Gregory V. Chudnovsky  Mark Clements
James Cloos  Jia Zhao Cong  Josh Cohen
Christophe Conil  Don Coppersmith  George Corliss
Robert Corless  Gary Cornell  Meino Cramer
Jeremy Du Croz  David Cyganski  Nathaniel Daly
Timothy Daly Sr.  Timothy Daly Jr.  James H. Davenport
David Day  James Demmel  Didier Deshommes
Michael Dewar  Jack Dongarra  Jean Della Dora
Gabriel Dos Reis  Claire DiCrescendo  Sam Dooley
Lionel Ducos  Iain Duff  Lee Duhem
Martin Dunstan  Brian Dupee  Dominique Duval
Robert Edwards  Heow Eide-Goodman  Lars Erickson
Richard Fateman  Bertfried Fauser  Stuart Feldman
John Fletcher  Brian Ford  Albrecht Fortenbacher
George Frances  Constantine Frangos  Timothy Freeman
Korrinn Fu  Marc Gaetano  Rudiger Gebauer
Van de Geijn  Kathy Gerber  Patricia Gianni
Gustavo Goertkin  Samantha Goldrich  Holger Gollan
Teresa Gomez-Diaz  Laureano Gonzalez-Vega  Stephen Gortler
Johannes Grabmeier  Matt Grayson  Klaus Ebbe Grue
James Griesmer  Vladimir Grinberg  Oswald Gschmitz
Ming Gu  Jocelyn Guidry  Gaetan Hache
Steve Hague  Satoshi Hamaguchi  Sven Hammarling
Mike Hansen  Richard Hanson  Richard Harke
Bill Hart  Vilya Harvey  Martin Hassner
Arthur S. Hathaway  Dan Hatton  Waldek Hebisch
Karl Hegbloom  Ralf Hemmecke  Henderson
Antoine Hersen  Roger House  Gernot Hueber
Pietro Iglio  Alejandro Jakubi  Richard Jenks
William Kahan  Kyriakos Kalorkoti  Kai Kaminski
# Contents

## 1 The Interpreter

## 2 The Fundamental Data Structures

- 2.1 Frames and the Interpreter Frame Ring .................................. 3
- 2.2 frame Command ............................................................... 3
  - frame man page .......................................................... 3
- 2.3 Data Structures ............................................................ 6
- 2.4 Frame Access Macros ......................................................... 7
  - defmacro frameName ...................................................... 7
  - defmacro frameInteractive ............................................ 8
  - defmacro frameIOIndex ................................................ 8
  - defmacro frameHiFiAccess ............................................. 8
  - defmacro frameHistList ................................................ 8
  - defmacro frameHistListLen ............................................ 9
  - defmacro frameHistListAct .......................................... 9
  - defmacro frameHistRecord ........................................... 9
  - defmacro frameHistoryTable ......................................... 10
  - defmacro frameExposureData ......................................... 10
- 2.5 Functions to manipulate frames ......................................... 10
  - The top level frame command ....................................... 10
  - The top level frame command handler ............................ 10
  - Initializing the Interpreter Frame Ring .......................... 12
  - Create a new, empty Interpreter Frame .......................... 12
  - Create a list of all of the frame names ......................... 13
  - Display the frame name list message ............................. 13
  - Collect the global variables into a Frame ...................... 14
  - Update global variables from the Current Frame ............... 15
  - Replace the current frame and update from the globals ...... 16
  - Get Named Frame Environment (aka Interactive) ............... 16
  - Find a Frame in the Frame Ring by Name ....................... 16
  - Change to the Named Interpreter Frame ........................ 17
  - Move to the next Interpreter Frame in Ring .................. 17
  - Move to the previous Interpreter Frame in Ring ............. 18
  - Add a New Interpreter Frame ....................................... 18
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import items from another frame</td>
<td>19</td>
</tr>
<tr>
<td>Close an Interpreter Frame</td>
<td>22</td>
</tr>
<tr>
<td>2.6 Global variables associated with the frame</td>
<td>23</td>
</tr>
<tr>
<td>defvar $InterpreterFrameRing</td>
<td>23</td>
</tr>
<tr>
<td>defvar $InterpreterFrameName</td>
<td>23</td>
</tr>
<tr>
<td>defvar $InteractiveFrame</td>
<td>23</td>
</tr>
<tr>
<td>defvar $IOindex</td>
<td>24</td>
</tr>
<tr>
<td>2.7 Interpreter Functions using Frames</td>
<td>24</td>
</tr>
<tr>
<td>3 The Message Mechanism</td>
<td>25</td>
</tr>
<tr>
<td>defvar $msgAlist</td>
<td>26</td>
</tr>
<tr>
<td>defvar $testingErrorPrefix</td>
<td>26</td>
</tr>
<tr>
<td>defvar $msgdbPrims</td>
<td>26</td>
</tr>
<tr>
<td>defvar $msgdbPunct</td>
<td>27</td>
</tr>
<tr>
<td>defvar $msgdbNoBlanksBeforeGroup</td>
<td>27</td>
</tr>
<tr>
<td>defvar $msgdbNoBlanksAfterGroup</td>
<td>27</td>
</tr>
<tr>
<td>defun Say a message using a keyed lookup</td>
<td>27</td>
</tr>
<tr>
<td>defun Handle msg formatting and print to file</td>
<td>28</td>
</tr>
<tr>
<td>defun Break a message into words</td>
<td>28</td>
</tr>
<tr>
<td>defun Write a msg into spadmsg.listing file</td>
<td>28</td>
</tr>
<tr>
<td>defun sayMSG</td>
<td>29</td>
</tr>
<tr>
<td>4 The History Mechanism</td>
<td>31</td>
</tr>
<tr>
<td>defvar $HiFiAccess</td>
<td>31</td>
</tr>
<tr>
<td>defvar $HistList</td>
<td>31</td>
</tr>
<tr>
<td>defvar $HistListLen</td>
<td>32</td>
</tr>
<tr>
<td>defvar $HistListAct</td>
<td>32</td>
</tr>
<tr>
<td>defvar $internalHistoryTable</td>
<td>32</td>
</tr>
<tr>
<td>defvar $HistRecord</td>
<td>32</td>
</tr>
<tr>
<td>defvar $historyFileType</td>
<td>33</td>
</tr>
<tr>
<td>5 The undo mechanism</td>
<td>35</td>
</tr>
<tr>
<td>5.1 Data Structures</td>
<td>35</td>
</tr>
<tr>
<td>5.2 Initial Undo Variables</td>
<td>35</td>
</tr>
<tr>
<td>defvar $frameRecord</td>
<td>35</td>
</tr>
<tr>
<td>defvar $previousBindings</td>
<td>36</td>
</tr>
<tr>
<td>defvar $reportundo</td>
<td>36</td>
</tr>
<tr>
<td>5.3 The undo functions</td>
<td>36</td>
</tr>
<tr>
<td>defun undo</td>
<td>36</td>
</tr>
<tr>
<td>defun undoSteps</td>
<td>37</td>
</tr>
<tr>
<td>defun undoSingleStep</td>
<td>38</td>
</tr>
<tr>
<td>defun undoLocalModemapHack</td>
<td>40</td>
</tr>
<tr>
<td>Remove undo lines from history write</td>
<td>41</td>
</tr>
<tr>
<td>defun reportUndo</td>
<td>43</td>
</tr>
<tr>
<td>Undo previous n commands</td>
<td>45</td>
</tr>
</tbody>
</table>
7 Exposure groups

7.1 Functions to manipulate exposure

7.2 Exposure Data Structures

8 The global variables

9 Starting Axiom

9.1 An Overview of a Simple Input

9.2 Parsing the input

10 Axiom Details

10.1 Variables Used

10.2 Data Structures

10.3 Functions

CONTENTS
CONTENTS

11 Handling Terminal Input 245
11.1 Streams 245
defvar curinstream 245
defvar curoutstream 245
defvar errorinstream 245
defvar erroroutstream 245
defvar *eof* 246
defvar *whitespace* 246
defvar $InteractiveMode 247
defvar $env 247
defvar $e 247
defvar $InteractiveMode 247
defvar $boot 248
$newspad 248
defvar $newspad 248
Top-level read-parse-eval-print loop 248
defun ncIntLoop 249
defvar $intTopLevel 249
defvar $intRestart 249
defun intloop 250
defvar $ncMsgList 250
defun SpadInterpretStream 250
defun GCL cmpnote function 251
defvar $newcompErrorCount 251
defvar $nopos 251
11.2 The Read-Eval-Print Loop 253
defun intloopReadConsole 253
11.3 Helper Functions 255
Get the value of an environment variable 255
defvar $intCoerceFailure 255
defvar $intSpadReader 255
defun InterpExecuteSpadSystemCommand 256
defun ExecuteInterpSystemCommand 256
defun substring 256
defun Handle Synonyms 257
defun Synonym File Reader 257
defun init-memory-config
Set spadroot to be the AXIOM shell variable
Does the string start with this prefix?
defun Interpret a line of lisp code
Get the current directory
Prepend the absolute path to a filename
Make the initial modemap frame
defun ncloopEscaped
defun intloopProcessString
defun ncloopParse
defun next
defun next1
defun incString
Call the garbage collector
defun reroot
defvar $current-directory
defun setCurrentLine
Show the Axiom prompt
defvar $frameAlist
defvar $frameNumber
defvar $currentFrameNum
defvar $EndServerSession
defvar $NeedToSignalSessionManager
defvar $sockBufferLength
READ-LINE in an Axiom server system
defun protectedEVAL
defvar $QuietCommand
defun executeQuietCommand
defun parseAndInterpret
defun parseFromString
defvar $interpOnly
defvar $minivectorNames
defvar $domPvar
defvar $compilingMap
defvar $instantRecord
defun processInteractive
defvar $ProcessInteractiveValue
defvar $HTCompanionWindowID
defun processInteractive1
defun interpretTopLevel
defvar $genValue
defun Type analyzes and evaluates expression x, returns object
defun Dispatcher for the type analysis routines
defvar $ThrowAwayMode
defun interpret2
defvar $runTestFlag
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>defvar $mkTestFlag</td>
<td>281</td>
</tr>
<tr>
<td>defun Result Output Printing</td>
<td>282</td>
</tr>
<tr>
<td>defun printStatisticsSummary</td>
<td>283</td>
</tr>
<tr>
<td>defun printStorage</td>
<td>283</td>
</tr>
<tr>
<td>defun printTypeAndTime</td>
<td>284</td>
</tr>
<tr>
<td>defun printAsTeX</td>
<td>285</td>
</tr>
<tr>
<td>defun sameUnionBranch</td>
<td>285</td>
</tr>
<tr>
<td>defun msgText</td>
<td>286</td>
</tr>
<tr>
<td>defun Right-justify the Type output</td>
<td>286</td>
</tr>
<tr>
<td>defun Destructively fix quotes in strings</td>
<td>287</td>
</tr>
<tr>
<td>Include a file into the stream</td>
<td>287</td>
</tr>
<tr>
<td>defun intloopInclude0</td>
<td>288</td>
</tr>
<tr>
<td>defun intloopProcess</td>
<td>288</td>
</tr>
<tr>
<td>defun intloopSpadProcess</td>
<td>289</td>
</tr>
<tr>
<td>defun intloopSpadProcess,interp</td>
<td>290</td>
</tr>
<tr>
<td>defun phParse</td>
<td>291</td>
</tr>
<tr>
<td>defun phIntReportMsgs</td>
<td>291</td>
</tr>
<tr>
<td>defun phInterpret</td>
<td>292</td>
</tr>
<tr>
<td>defun intInterpreterPform</td>
<td>292</td>
</tr>
<tr>
<td>defun zeroOneTran</td>
<td>292</td>
</tr>
<tr>
<td>defun ncConversationPhase</td>
<td>292</td>
</tr>
<tr>
<td>defun ncConversationPhase,wrapup</td>
<td>293</td>
</tr>
<tr>
<td>defun ncError</td>
<td>293</td>
</tr>
<tr>
<td>defun intloopEchoParse</td>
<td>294</td>
</tr>
<tr>
<td>defun ncloopPrintLines</td>
<td>294</td>
</tr>
<tr>
<td>defun mkLineList</td>
<td>295</td>
</tr>
<tr>
<td>defun nonBlank</td>
<td>295</td>
</tr>
<tr>
<td>defun ncloopDQlines</td>
<td>296</td>
</tr>
<tr>
<td>defun poGlobalLinePosn</td>
<td>296</td>
</tr>
<tr>
<td>defun streamChop</td>
<td>297</td>
</tr>
<tr>
<td>defun ncloopInclude0</td>
<td>297</td>
</tr>
<tr>
<td>defun incStream</td>
<td>298</td>
</tr>
<tr>
<td>defun incRenumber</td>
<td>298</td>
</tr>
<tr>
<td>defun incZip</td>
<td>298</td>
</tr>
<tr>
<td>defun incZip1</td>
<td>299</td>
</tr>
<tr>
<td>defun incIgen</td>
<td>299</td>
</tr>
<tr>
<td>defun incIgen1</td>
<td>300</td>
</tr>
<tr>
<td>defun incRenumberLine</td>
<td>300</td>
</tr>
<tr>
<td>defun incRenumberItem</td>
<td>300</td>
</tr>
<tr>
<td>defun incHandleMessage</td>
<td>301</td>
</tr>
<tr>
<td>defun incLude</td>
<td>301</td>
</tr>
<tr>
<td>defmacro Rest</td>
<td>302</td>
</tr>
<tr>
<td>defvar Top</td>
<td>302</td>
</tr>
<tr>
<td>defvar IfSkipToEnd</td>
<td>302</td>
</tr>
<tr>
<td>defvar IfKeepPart</td>
<td>302</td>
</tr>
<tr>
<td>defvar IfSkipPart</td>
<td>302</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>defun xlPrematureFin</td>
<td>321</td>
</tr>
<tr>
<td>defun incmsgPrematureFin</td>
<td>321</td>
</tr>
<tr>
<td>defun assertCond</td>
<td>321</td>
</tr>
<tr>
<td>defun xIfSyntax</td>
<td>322</td>
</tr>
<tr>
<td>defun incmsgxIfSyntax</td>
<td>322</td>
</tr>
<tr>
<td>defun xlIfBug</td>
<td>323</td>
</tr>
<tr>
<td>defun incmsgxIfBug</td>
<td>323</td>
</tr>
<tr>
<td>defun xIcmdBug</td>
<td>323</td>
</tr>
<tr>
<td>defun incmsgxIcmdBug</td>
<td>323</td>
</tr>
<tr>
<td>defvar incCommands</td>
<td>324</td>
</tr>
<tr>
<td>defvar $pfMacros</td>
<td>324</td>
</tr>
<tr>
<td>defun incClassify</td>
<td>324</td>
</tr>
<tr>
<td>defun incCommand?</td>
<td>326</td>
</tr>
<tr>
<td>defun incPrefix?</td>
<td>326</td>
</tr>
<tr>
<td>defun incCommandTail</td>
<td>327</td>
</tr>
<tr>
<td>defun incDrop</td>
<td>327</td>
</tr>
<tr>
<td>defun incFname</td>
<td>327</td>
</tr>
<tr>
<td>defun incFileInput</td>
<td>327</td>
</tr>
<tr>
<td>defun incConsoleInput</td>
<td>328</td>
</tr>
<tr>
<td>defun incNConsoles</td>
<td>328</td>
</tr>
<tr>
<td>defun incActive?</td>
<td>328</td>
</tr>
<tr>
<td>defun incRgen</td>
<td>329</td>
</tr>
<tr>
<td>defun Delay</td>
<td>329</td>
</tr>
<tr>
<td>defvar StreamNil</td>
<td>329</td>
</tr>
<tr>
<td>defun incRgen1</td>
<td>330</td>
</tr>
</tbody>
</table>

### 12 The Token Scanner 331

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>defvar scanKeyWords</td>
<td>331</td>
</tr>
<tr>
<td>defvar infgeneric</td>
<td>333</td>
</tr>
<tr>
<td>defun lineoftoks</td>
<td>334</td>
</tr>
<tr>
<td>defun nextline</td>
<td>336</td>
</tr>
<tr>
<td>defun scanIgnoreLine</td>
<td>336</td>
</tr>
<tr>
<td>defun constoken</td>
<td>337</td>
</tr>
<tr>
<td>defun scanToken</td>
<td>337</td>
</tr>
<tr>
<td>defun lfid</td>
<td>338</td>
</tr>
<tr>
<td>defun Is it a ++ comment?</td>
<td>339</td>
</tr>
<tr>
<td>defun scanComment</td>
<td>339</td>
</tr>
<tr>
<td>defun lfcomment</td>
<td>340</td>
</tr>
<tr>
<td>defun Is it a – comment?</td>
<td>340</td>
</tr>
<tr>
<td>defun scanNegComment</td>
<td>340</td>
</tr>
<tr>
<td>defun lfnegcomment</td>
<td>341</td>
</tr>
<tr>
<td>defun punctuation?</td>
<td>341</td>
</tr>
<tr>
<td>defun scanPunct</td>
<td>341</td>
</tr>
<tr>
<td>defun subMatch</td>
<td>342</td>
</tr>
<tr>
<td>defun substringMatch</td>
<td>342</td>
</tr>
<tr>
<td>defun scanKeyTr</td>
<td>343</td>
</tr>
</tbody>
</table>
defun keyword .................................................. 344
defun keyword? ................................................ 344
defun scanPossFloat ........................................... 344
defun digit? ................................................... 345
defun lfkey ..................................................... 345
defun spleI ..................................................... 346
defun spleI1 .................................................... 346
defun scanEsc .................................................. 346
defvar scanCloser .............................................. 348
defun scanCloser? ............................................. 349
defun scanWord ............................................... 349
defun scanExponent .......................................... 349
defun lffloat .................................................. 351
defmacro idChar? .............................................. 351
defun scanW .................................................... 351
defun posend .................................................. 352
defun scanSpace ............................................... 352
defun lfspaces ............................................... 353
defun scanString ............................................. 353
defun lstring .................................................. 353
defun scanS .................................................... 354
defun scanTransform .......................................... 355
defun scanNumber ............................................. 355
defun rdigit? .................................................. 356
defun lfinteger ............................................... 357
defun lfinteger ............................................... 357
defun scanCheckRadix ......................................... 357
defun scanEscape ............................................. 358
defun scanError ............................................... 358
defun lerror ................................................... 359
defvar scanKeyTable ........................................... 359
defun scanKeyTableCons ...................................... 359
defvar scanDict ............................................... 360
defun scanDictCons .......................................... 360
defun scanInsert ............................................. 361
defvar scanPun ............................................... 362
defun scanPunCons ............................................ 362

13 Input Stream Parser ........................................ 365
defun Input Stream Parser ..................................... 365
defun npItem .................................................. 366
defun npItem1 .................................................. 366
defun npFirstTok .............................................. 367
defun Push one item onto $stack ............................. 367
defun Pop one item off $stack ................................ 368
defun Pop the second item off $stack ....................... 368
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>defun Pop the third item off $stack</td>
<td>368</td>
</tr>
<tr>
<td>defun npQualDef</td>
<td>369</td>
</tr>
<tr>
<td>defun Advance over a keyword</td>
<td>369</td>
</tr>
<tr>
<td>defun Advance the input stream</td>
<td>369</td>
</tr>
<tr>
<td>defun npComma</td>
<td>370</td>
</tr>
<tr>
<td>defun npTuple</td>
<td>370</td>
</tr>
<tr>
<td>defun npCommaBackSet</td>
<td>370</td>
</tr>
<tr>
<td>defun npQualifiedDefinition</td>
<td>371</td>
</tr>
<tr>
<td>defun npQualified</td>
<td>371</td>
</tr>
<tr>
<td>defun npDefinitionOrStatement</td>
<td>372</td>
</tr>
<tr>
<td>defun npBackTrack</td>
<td>372</td>
</tr>
<tr>
<td>defun npGives</td>
<td>372</td>
</tr>
<tr>
<td>defun npLambda</td>
<td>373</td>
</tr>
<tr>
<td>defun npType</td>
<td>373</td>
</tr>
<tr>
<td>defun npMatch</td>
<td>374</td>
</tr>
<tr>
<td>defun npSuch</td>
<td>374</td>
</tr>
<tr>
<td>defun npWith</td>
<td>374</td>
</tr>
<tr>
<td>defun npCompMissing</td>
<td>375</td>
</tr>
<tr>
<td>defun npMissing</td>
<td>375</td>
</tr>
<tr>
<td>defun npRestore</td>
<td>376</td>
</tr>
<tr>
<td>defun Peek for keyword s, no advance of token stream</td>
<td>376</td>
</tr>
<tr>
<td>defun npCategoryL</td>
<td>377</td>
</tr>
<tr>
<td>defun npCategory</td>
<td>377</td>
</tr>
<tr>
<td>defun npSCategory</td>
<td>377</td>
</tr>
<tr>
<td>defun npSignature</td>
<td>378</td>
</tr>
<tr>
<td>defun npSigItemlist</td>
<td>378</td>
</tr>
<tr>
<td>defun npListing</td>
<td>379</td>
</tr>
<tr>
<td>defun Always produces a list, fn is applied to it</td>
<td>379</td>
</tr>
<tr>
<td>defun npSigItem</td>
<td>380</td>
</tr>
<tr>
<td>defun npTypeVariable</td>
<td>380</td>
</tr>
<tr>
<td>defun npSignatureDefinee</td>
<td>381</td>
</tr>
<tr>
<td>defun npTypeVariablelist</td>
<td>381</td>
</tr>
<tr>
<td>defun npSigDecl</td>
<td>381</td>
</tr>
<tr>
<td>defun npPrimary</td>
<td>382</td>
</tr>
<tr>
<td>defun npPrimary2</td>
<td>382</td>
</tr>
<tr>
<td>defun npADD</td>
<td>382</td>
</tr>
<tr>
<td>defun npAdd</td>
<td>383</td>
</tr>
<tr>
<td>defun npAtom2</td>
<td>384</td>
</tr>
<tr>
<td>defun npInfixOperator</td>
<td>384</td>
</tr>
<tr>
<td>defun npInfixOp</td>
<td>385</td>
</tr>
<tr>
<td>defun npPrefixColon</td>
<td>385</td>
</tr>
<tr>
<td>defun npApplication</td>
<td>386</td>
</tr>
<tr>
<td>defun npDotted</td>
<td>386</td>
</tr>
<tr>
<td>defun npAnyNo</td>
<td>387</td>
</tr>
<tr>
<td>defun npSelector</td>
<td>387</td>
</tr>
<tr>
<td>defun npApplication2</td>
<td>387</td>
</tr>
<tr>
<td>defun</td>
<td>npDefinition</td>
</tr>
<tr>
<td>------</td>
<td>--------------</td>
</tr>
<tr>
<td>defun</td>
<td>npDollar</td>
</tr>
<tr>
<td>defun</td>
<td>npConstTok</td>
</tr>
<tr>
<td>defun</td>
<td>npBDefinition</td>
</tr>
<tr>
<td>defun</td>
<td>npBracketed</td>
</tr>
<tr>
<td>defun</td>
<td>npParened</td>
</tr>
<tr>
<td>defun</td>
<td>npBracked</td>
</tr>
<tr>
<td>defun</td>
<td>npBraced</td>
</tr>
<tr>
<td>defun</td>
<td>npSingleBared</td>
</tr>
<tr>
<td>defun</td>
<td>npDefn</td>
</tr>
<tr>
<td>defun</td>
<td>npDef</td>
</tr>
<tr>
<td>defun</td>
<td>npBPileDefinition</td>
</tr>
<tr>
<td>defun</td>
<td>npPileBracketed</td>
</tr>
<tr>
<td>defun</td>
<td>npPileDefinitionlist</td>
</tr>
<tr>
<td>defun</td>
<td>npListAndRecover</td>
</tr>
<tr>
<td>defun</td>
<td>npRecoverTrap</td>
</tr>
<tr>
<td>defun</td>
<td>npMoveTo</td>
</tr>
<tr>
<td>defun</td>
<td>sylIgnoredFromTo</td>
</tr>
<tr>
<td>defun</td>
<td>syGeneralErrorHere</td>
</tr>
<tr>
<td>defun</td>
<td>sySpecificErrorHere</td>
</tr>
<tr>
<td>defun</td>
<td>sySpecificErrorAtToken</td>
</tr>
<tr>
<td>defun</td>
<td>npDefinitionlist</td>
</tr>
<tr>
<td>defun</td>
<td>npSemiListing</td>
</tr>
<tr>
<td>defun</td>
<td>npSemiBackSet</td>
</tr>
<tr>
<td>defun</td>
<td>npRule</td>
</tr>
<tr>
<td>defun</td>
<td>npSingleRule</td>
</tr>
<tr>
<td>defun</td>
<td>npDefTail</td>
</tr>
<tr>
<td>defun</td>
<td>npDefaultValue</td>
</tr>
<tr>
<td>defun</td>
<td>npWConditional</td>
</tr>
<tr>
<td>defun</td>
<td>npConditional</td>
</tr>
<tr>
<td>defun</td>
<td>npElse</td>
</tr>
<tr>
<td>defun</td>
<td>npBacksetElse</td>
</tr>
<tr>
<td>defun</td>
<td>npLogical</td>
</tr>
<tr>
<td>defun</td>
<td>npDisjand</td>
</tr>
<tr>
<td>defun</td>
<td>npDiscrim</td>
</tr>
<tr>
<td>defun</td>
<td>npQuiver</td>
</tr>
<tr>
<td>defun</td>
<td>npRelation</td>
</tr>
<tr>
<td>defun</td>
<td>npSynthetic</td>
</tr>
<tr>
<td>defun</td>
<td>npBy</td>
</tr>
<tr>
<td>defun</td>
<td>npSegment</td>
</tr>
<tr>
<td>defun</td>
<td>npArith</td>
</tr>
<tr>
<td>defun</td>
<td>npSum</td>
</tr>
<tr>
<td>defun</td>
<td>npTerm</td>
</tr>
<tr>
<td>defun</td>
<td>npRemainder</td>
</tr>
<tr>
<td>defun</td>
<td>npProduct</td>
</tr>
</tbody>
</table>
defun pf0FlattenSyntacticTuple .................................................. 467
defun pSourcePosition ............................................................. 468
defun Convert a Sequence node to a list ........................................ 468
defun pfSpread ........................................................................... 469
defun Deconstruct nodes to lists .................................................... 469
defun pfCheckMacroOut ............................................................... 470
defun pfCheckArg ......................................................................... 471
defun pfCheckId ............................................................................ 471
defun pfFlattenApp ................................................................. 471
defun pfCollect1? ................................................................. 472
defun pfCollectVariable1 .............................................................. 472
defun pfPushMacroBody ............................................................... 473
defun pfSourceStok ................................................................. 473
defun pfTransformArg .............................................................. 474
defun pfTaggedToTyped1 .............................................................. 474
defun pfSuch ................................................................. 474

14.3 Special Nodes ................................................................. 475
defun Create a Listof node .......................................................... 475
defun pfNothing ................................................................. 475
defun Is this a Nothing node? ...................................................... 475

14.4 Leaves ................................................................. 476
defun Create a Document node ...................................................... 476
defun Construct an Id node .......................................................... 476
defun Is this an Id node? .............................................................. 476
defun Construct an Id leaf node .................................................... 476
defun Return the Id part .............................................................. 477
defun Construct a Leaf node .......................................................... 477
defun Is this a leaf node? .............................................................. 477
defun Return the token position of a leaf node ................................ 478
defun Return the Leaf Token .......................................................... 478
defun Is this a Literal node? .......................................................... 478
defun Create a LiteralClass node .................................................... 478
defun Return the LiteralString ...................................................... 479
defun Return the parts of a tree node .............................................. 479
defun Return the argument unchanged ........................................... 479
defun pfPushBody ................................................................. 479
defun An S-expression which people can read. ................................ 480
defun Create a human readable S-expression .................................... 480
defun Construct a Symbol or Expression node .................................. 481
defun Construct a Symbol leaf node ................................................. 481
defun Is this a Symbol node? .......................................................... 482
defun Return the Symbol part .......................................................... 482

14.5 Trees ................................................................. 482
defun Construct a tree node .......................................................... 482
defun Construct an Add node .......................................................... 482
defun Construct an And node .......................................................... 483
defun pfAttribute .............................................. 483
defun Return an Application node ............................. 483
defun Return the Arg part of an Application node .......... 484
defun Return the Op part of an Application node .......... 484
defun Is this an And node? .................................. 484
defun Return the Left part of an And node .................... 484
defun Return the Right part of an And node .................... 485
defun Flatten a list of lists .................................. 485
defun Is this an Application node? ............................ 485
defun Create an Assign node .................................. 485
defun Is this an Assign node? ................................ 486
defun Return the parts of an LhsItem of an Assign node .... 486
defun Return the LhsItem of an Assign node .................... 486
defun Return the RHS of an Assign node ........................ 486
defun Construct an application node for a brace .......... 487
defun Construct an Application node for brace-bars ......... 487
defun Construct an Application node for a bracket ........ 487
defun Construct an Application node for bracket-bars ...... 487
defun Create a Break node ..................................... 488
defun Is this a Break node? .................................... 488
defun Return the From part of a Break node ................... 488
defun Construct a Coerceto node ................................ 489
defun Is this a CoerceTo node? ................................ 489
defun Return the Type part of a CoerceTo node .............. 489
defun Return the Body of a CoerceTo node.................... 489
defun Return the Iterators of a Collect node ............... 490
defun Create a Collect node .................................... 490
defun Is this a Collect node? .................................. 490
defun pfDefinition .............................................. 491
defun Return the Lhs of a Definition node ..................... 491
defun Return the Rhs of a Definition node .................... 491
defun Is this a Definition node? ............................... 491
defun Return the parts of a Definition node ................. 492
defun Create a Do node ......................................... 492
defun Is this a Do node? ........................................ 492
defun Return the Body of a Do node ........................... 492
defun Construct a Sequence node ................................ 493
defun Create an Exit node ...................................... 493
defun Is this an Exit node? ..................................... 493
defun Return the Cond part of an Exit ......................... 493
defun Return the Expression part of an Exit .................. 494
defun Create an Export node .................................... 494
defun Construct an Expression leaf node ........................ 494
defun pfFirst .................................................... 494
defun Create an Application Fix node .......................... 495
CONTENTS

defun Create a Free node  ........................................  495
defun Is this a Free node?  .......................................  495
defun Return the parts of the Items of a Free node  ..........  496
defun Return the Items of a Free node  ..........................  496
defun Construct a ForIn node  .....................................  496
defun Is this a ForIn node?  .......................................  496
defun Return all the parts of the LHS of a ForIn node  ......  497
defun Return the LHS part of a ForIn node  ......................  497
defun Return the Whole part of a ForIn node  ..................  497
defun pfFromDom  ..................................................  497
defun Construct a Fromdom node ....................................  498
defun Is this a Fromdom node?  ....................................  498
defun Return the What part of a Fromdom node  ................  498
defun Return the Domain part of a Fromdom node  ..............  499
defun Construct a Hide node  ......................................  499
defun pfIf  .........................................................  499
defun Is this an If node?  .........................................  499
defun Return the Cond part of an If  .............................  500
defun Return the Then part of an If  .............................  500
defun pfIfThenOnly  ................................................  500
defun Return the Else part of an If  .............................  500
defun Construct an Import node  ....................................  501
defun Construct an Iterate node  ..................................  501
defun Is this an Iterate node?  ....................................  501
defun Handle an infix application  ................................  501
defun Create an Inline node  .......................................  502
defun pfLam  .......................................................  502
defun pfLambda  ....................................................  503
defun Return the Body part of a Lambda node  .................  503
defun Return the Rets part of a Lambda node  ...................  503
defun Is this a Lambda node?  .....................................  503
defun Return the Args part of a Lambda node  ...................  504
defun Return the Args of a Lambda Node  ..........................  504
defun Construct a Local node  .....................................  504
defun Is this a Local node?  .......................................  504
defun Return the parts of Items of a Local node  ..............  505
defun Return the Items of a Local node  ........................  505
defun Construct a Loop node  .....................................  505
defun pfLoop1  .....................................................  505
defun Is this a Loop node?  .......................................  506
defun Return the Iterators of a Loop node  ......................  506
defun pf0LoopIterators  ............................................  506
defun pfLp  .........................................................  506
defun Create a Macro node  ........................................  507
defun Is this a Macro node?  ......................................  507
defun Return the Lhs of a Macro node  ...........................  507
defun Return the Rhs of a Macro node .......................... 507
defun Construct an MLambda node ................................. 508
defun Is this an MLambda node? .................................. 508
defun Return the Args of an MLambda ............................. 508
defun Return the parts of an MLambda argument ................. 508
defun pfMLambdaBody .................................................. 509
defun Is this a Not node? ............................................ 509
defun Return the Arg part of a Not node .......................... 509
defun Construct a NoValue node .................................... 509
defun Is this a NoValue node? .................................... 510
defun Return the Expr part of a NoValue node .................. 510
defun Construct an Or node ......................................... 510
defun Is this an Or node? .......................................... 510
defun Return the Left part of an Or node ......................... 511
defun Return the Right part of an Or node ....................... 511
defun Return the part of a parenthesised expression ........... 511
defun pfPretend .......................................................... 511
defun Is this a Pretend node? ..................................... 512
defun Return the Expression part of a Pretend node ............ 512
defun Return the Type part of a Pretend node ................... 512
defun Construct a QualType node .................................. 512
defun Construct a Restrict node ................................... 513
defun Is this a Restrict node? .................................... 513
defun Return the Expr part of a Restrict node .................. 513
defun Return the Type part of a Restrict node ................. 513
defun Construct a RetractTo node ................................ 514
defun Construct a Return node ..................................... 514
defun Is this a Return node? ..................................... 514
defun Return the Expr part of a Return node .................... 514
defun pfReturnNoName ................................................ 515
defun Construct a ReturnTyped node ............................... 515
defun Construct a Rule node ....................................... 515
defun Return the Lhs of a Rule node ............................... 516
defun Return the Rhs of a Rule node ............................... 516
defun Is this a Rule node? .......................................... 516
defun pfSecond .......................................................... 516
defun Construct a Sequence node ................................ 517
defun Return the Args of a Sequence node ....................... 517
defun Is this a Sequence node? .................................... 517
defun Return the parts of the Args of a Sequence node ......... 517
defun Create a Suchthat node ...................................... 518
defun Is this a SuchThat node? .................................... 518
defun Return the Cond part of a SuchThat node ................ 518
defun Create a Tagged node ........................................ 518
defun Is this a Tagged node? ..................................... 519
defun Return the Expression portion of a Tagged node ......... 519
defun Return the Tag of a Tagged node ........................................ 519
defun pfTaggedToTyped ..................................................... 519
defun pfTweakIf ........................................................................ 520
defun Construct a Typed node .................................................. 520
defun Is this a Typed node? ................................................... 521
defun Return the Type of a Typed node ...................................... 521
defun Return the Id of a Typed node ......................................... 521
defun Construct a Typing node ................................................ 521
defun Return a Tuple node ..................................................... 522
defun Return a Tuple from a List ............................................. 522
defun Is this a Tuple node? .................................................... 522
defun Return the Parts of a Tuple node ..................................... 523
defun Return the parts of a Tuple ............................................... 523
defun Return a list from a Sequence node ................................... 523
defun The comment is attached to all signatures .......................... 523
defun Construct a WDeclare node ............................................. 524
defun Construct a Where node ................................................ 524
defun Is this a Where node? ................................................... 524
defun Return the parts of the Context of a Where node .................. 525
defun Return the Context of a Where node ................................. 525
defun Return the Expr part of a Where node ............................... 525
defun Construct a While node ................................................ 525
defun Is this a While node? .................................................... 526
defun Return the Cond part of a While node ................................. 526
defun Construct a With node .................................................. 526
defun Create a Wrong node .................................................... 526
defun Is this a Wrong node? ................................................... 527

15 Pftree to s-expression translation ........................................ 529
defun Pftree to s-expression translation .................................... 529
defun Pftree to s-expression translation inner function ................. 530
defun Convert a Literal to an S-expression ................................. 531
defun Convert a float to an S-expression .................................... 534
defun Change an Application node to an S-expression ..................... 535
defun Convert a SuchThat node to an S-expression ........................ 537
defun pfOp2Sex ...................................................................... 538
defun pmDontQuote? ................................................................ 539
defun hasOptArgs? .................................................................. 539
defun Convert a Sequence node to an S-expression ........................ 540
defun pSequence2Sex0 ........................................................... 540
defun Convert a loop node to an S-expression ............................... 541
defun Change a Collect node to an S-expression ............................ 544
defun Convert a Definition node to an S-expression ....................... 545
defun Convert a Lambda node to an S-expression ......................... 546
defun pfCollectArgTran .......................................................... 547
defun Convert a Lambda node to an S-expression ........................... 548
defun Line element 4: Is it a filename? ............................... 569
defun Line element 4: Is it a filename? ............................... 569
defun Line element 4: Get filename ................................. 569

19.2 Messages ......................................................... 569
defun msgCreate ....................................................... 569
defmacro getMsgPosTagOb .............................................. 570
defmacro getMsgKey .................................................... 570
defmacro getMsgArgL ................................................... 571
defmacro getMsgPrefix ................................................ 571
defmacro setMsgPrefix ............................................... 571
defmacro getMsgText .................................................. 571
defmacro setMsgText .................................................. 571
defmacro getMsgPrefix? .............................................. 572
defmacro getMsgTag ................................................... 572
defmacro getMsgTag? ................................................... 572
defmacro line? ......................................................... 573
defmacro leader? ....................................................... 573
defmacro toScreen? .................................................... 573
defun ncSoftError ...................................................... 574
defun ncHardError ..................................................... 574
defun desiredMsg ....................................................... 575
defun processKeyedError .............................................. 575
defun msgOutputter .................................................... 575
defun listOutputter ..................................................... 576
defun getStFromMsg .................................................... 576
defvar $preLength ...................................................... 577
defun getPreStL ........................................................ 577
defun getPosStL ........................................................ 578
defun ppos ............................................................. 579
defun remFile .......................................................... 579
defun showMsgPos? ...................................................... 580
defvar $imPrGuys ....................................................... 580
defun msgImPr? .......................................................... 580
defun getMsgCatAttr .................................................... 580
defun getMsgPos ........................................................ 581
defun getMsgFTTag? ..................................................... 581
defun decideHowMuch ................................................. 581
defun poNopos? ......................................................... 582
defun poPosImmediate? ................................................. 582
defun poFileName ....................................................... 582
defun poGetLineObject ................................................ 583
defun poLinePosn ........................................................ 583
defun listDecideHowMuch ............................................. 583
defun remLine .......................................................... 584
defun getMsgKey? ....................................................... 584
defun tabbing ........................................................... 584
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>defvar $toWhereGuys</td>
<td>585</td>
</tr>
<tr>
<td>defun getMsgToWhere</td>
<td>585</td>
</tr>
<tr>
<td>defun toFile?</td>
<td>585</td>
</tr>
<tr>
<td>defun alreadyOpened?</td>
<td>585</td>
</tr>
<tr>
<td>defun setMsgForcedAttrList</td>
<td>586</td>
</tr>
<tr>
<td>defun setMsgForcedAttr</td>
<td>586</td>
</tr>
<tr>
<td>defvar $attrCats</td>
<td>586</td>
</tr>
<tr>
<td>defun whichCat</td>
<td>587</td>
</tr>
<tr>
<td>defun setMsgCatlessAttr</td>
<td>587</td>
</tr>
<tr>
<td>defun putDatabaseStuff</td>
<td>587</td>
</tr>
<tr>
<td>defun getMsgInfoFromKey</td>
<td>588</td>
</tr>
<tr>
<td>defun setMsgUnforcedAttrList</td>
<td>588</td>
</tr>
<tr>
<td>defun setMsgUnforcedAttr</td>
<td>589</td>
</tr>
<tr>
<td>defvar SimPrTagGuys</td>
<td>589</td>
</tr>
<tr>
<td>defun initImPr</td>
<td>589</td>
</tr>
<tr>
<td>defun initToWhere</td>
<td>590</td>
</tr>
<tr>
<td>defun Report a bug in the compiler</td>
<td>590</td>
</tr>
<tr>
<td>defun processMsgList</td>
<td>591</td>
</tr>
<tr>
<td>defun erMsgSort</td>
<td>591</td>
</tr>
<tr>
<td>defun erMsgCompare</td>
<td>592</td>
</tr>
<tr>
<td>defun compareposns</td>
<td>592</td>
</tr>
<tr>
<td>defun erMsgSep</td>
<td>592</td>
</tr>
<tr>
<td>defun makeMsgFromLine</td>
<td>593</td>
</tr>
<tr>
<td>defun rep</td>
<td>593</td>
</tr>
<tr>
<td>defun getLinePos</td>
<td>594</td>
</tr>
<tr>
<td>defun getLineText</td>
<td>594</td>
</tr>
<tr>
<td>defun queueUpErrors</td>
<td>594</td>
</tr>
<tr>
<td>defun thisPosIsLess</td>
<td>596</td>
</tr>
<tr>
<td>defun thisPosIsEqual</td>
<td>596</td>
</tr>
<tr>
<td>defun redundant</td>
<td>596</td>
</tr>
<tr>
<td>defvar $repGuys</td>
<td>597</td>
</tr>
<tr>
<td>defun msgNoRep?</td>
<td>597</td>
</tr>
<tr>
<td>defun sameMsg?</td>
<td>598</td>
</tr>
<tr>
<td>defun processChPosesForOneLine</td>
<td>598</td>
</tr>
<tr>
<td>defun poCharPosn</td>
<td>599</td>
</tr>
<tr>
<td>defun makeLeaderMsg</td>
<td>599</td>
</tr>
<tr>
<td>defun posPoints</td>
<td>600</td>
</tr>
<tr>
<td>defun getMsgPos2</td>
<td>600</td>
</tr>
<tr>
<td>defun insertPos</td>
<td>601</td>
</tr>
<tr>
<td>defun putFTText</td>
<td>601</td>
</tr>
<tr>
<td>defun From</td>
<td>602</td>
</tr>
<tr>
<td>defun To</td>
<td>602</td>
</tr>
<tr>
<td>defun FromTo</td>
<td>602</td>
</tr>
</tbody>
</table>
# CONTENTS

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>defvar $FontTable</td>
<td>643</td>
</tr>
<tr>
<td>defvar $Integer</td>
<td>643</td>
</tr>
<tr>
<td>defvar $ComplexInteger</td>
<td>643</td>
</tr>
<tr>
<td>defvar $Mode</td>
<td>643</td>
</tr>
<tr>
<td>defvar $NegativeInteger</td>
<td>643</td>
</tr>
<tr>
<td>defvar $NonNegativeInteger</td>
<td>644</td>
</tr>
<tr>
<td>defvar $NonPositiveInteger</td>
<td>644</td>
</tr>
<tr>
<td>defvar $PositiveInteger</td>
<td>644</td>
</tr>
<tr>
<td>defvar $RationalNumber</td>
<td>644</td>
</tr>
<tr>
<td>defvar $String</td>
<td>645</td>
</tr>
<tr>
<td>defvar $StringCategory</td>
<td>645</td>
</tr>
<tr>
<td>defvar $Symbol</td>
<td>645</td>
</tr>
<tr>
<td>defvar $Void</td>
<td>645</td>
</tr>
<tr>
<td>defvar $QuotientField</td>
<td>645</td>
</tr>
<tr>
<td>defvar $FunctionalExpression</td>
<td>646</td>
</tr>
<tr>
<td>defvar $defaultFunctionTargets</td>
<td>646</td>
</tr>
<tr>
<td>defvar $SmallInteger</td>
<td>646</td>
</tr>
<tr>
<td>defvar $SingleFloat</td>
<td>646</td>
</tr>
<tr>
<td>defvar $DoubleFloat</td>
<td>647</td>
</tr>
<tr>
<td>defvar $SingleInteger</td>
<td>647</td>
</tr>
</tbody>
</table>

## 23 Function Selection 649

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>defun ofCategory</td>
<td>649</td>
</tr>
<tr>
<td>defun isPartialMode</td>
<td>650</td>
</tr>
<tr>
<td>defun hasCaty</td>
<td>650</td>
</tr>
<tr>
<td>defun domArg</td>
<td>652</td>
</tr>
<tr>
<td>defun domArg2</td>
<td>652</td>
</tr>
<tr>
<td>defun hasSig</td>
<td>653</td>
</tr>
<tr>
<td>defun hasAtt</td>
<td>654</td>
</tr>
<tr>
<td>defun hasSigAnd</td>
<td>655</td>
</tr>
<tr>
<td>defun hasSigOr</td>
<td>656</td>
</tr>
<tr>
<td>defun hasAttSig</td>
<td>656</td>
</tr>
<tr>
<td>defun hasCate1</td>
<td>657</td>
</tr>
<tr>
<td>defun hasCatExpression</td>
<td>657</td>
</tr>
<tr>
<td>defun unifyStruct</td>
<td>658</td>
</tr>
<tr>
<td>defun unifyStructVar</td>
<td>659</td>
</tr>
<tr>
<td>defun containsVars</td>
<td>661</td>
</tr>
<tr>
<td>defun isPatternVar</td>
<td>661</td>
</tr>
<tr>
<td>defun containsVars1</td>
<td>661</td>
</tr>
<tr>
<td>defun hasCaty1</td>
<td>662</td>
</tr>
<tr>
<td>defun mkDomPvar</td>
<td>663</td>
</tr>
<tr>
<td>defun hasCate</td>
<td>663</td>
</tr>
<tr>
<td>defun constructSubst</td>
<td>664</td>
</tr>
<tr>
<td>defun hasCateSpecial</td>
<td>665</td>
</tr>
<tr>
<td>defun hasCateSpecialNew</td>
<td>666</td>
</tr>
<tr>
<td>defun defaultTargetFE</td>
<td>667</td>
</tr>
</tbody>
</table>
24 Coercions

defun isEqualOrSubDomain ........................................... 668

defun coerceInteractive ............................................. 672
defun coerceInt ..................................................... 673
defun coerceInt0 .................................................... 674
defun coerceInt1 .................................................... 675
defun coerceByFunction ............................................... 680
defun coerceIntTower .................................................. 682
defun coerceIntTest ................................................... 684
defun coerceConvertMmSelection ...................................... 684
defun hasCorrectTarget ............................................... 685
defun coerceIntPermute ................................................ 685
defun computeTTTTranspositions ..................................... 686
defun permuteToOrder .................................................. 687
defun decomposeTypeIntoTower ....................................... 688
defun reassembleTowerIntoType ...................................... 689
defun coerceIntCommute .............................................. 689
defun coerceCommuteTest ............................................... 690
defun coerceIntTableOrFunction ..................................... 690
defun coerceByTable ................................................... 691
defun catchCoerceFailure ............................................. 692
defun coerceIntSpecial ................................................ 692
defun coerceIntByMap .................................................. 693
defun coerceIntByMapInner ............................................ 694
defun coerceOrThrowFailure .......................................... 695
defun coercionFailure .................................................. 695
defun valueArgsEqual? .................................................. 695
defun algEqual .......................................................... 696
defun coerceIntFromUnion ............................................. 697
defun coerceInt2Union .................................................. 697
defun coerceBranch2Union ............................................. 698
defun coerceIntAlgebraicConstant .................................... 698
defun getConstantFromDomain ........................................ 699
defun compareTypeLists .............................................. 700
defun coercionIntX ..................................................... 700
defun coerceSubDomain ............................................... 700
defun coerceImmediateSubDomain ..................................... 701
defun getSubDomainPredicate ......................................... 701
defun absolutelyCanCoerceByCheating ................................ 702
defun coerceOrRetract .................................................. 703
defun retract2Specialization ......................................... 704
defun coerceUnion2Branch ............................................ 707
defun stripUnionTags ................................................... 708
defun evalSharpOne ..................................................... 708
defun retractUnderDomain ............................................. 709


\textbf{25 System Command Handling} \quad 713

25.1 Variables Used \quad 715
\begin{itemize}
  \item defvar $systemCommands \quad 715
  \item defvar $syscommands \quad 716
  \item defvar $noParseCommands \quad 716
\end{itemize}

25.2 Functions \quad 717
\begin{itemize}
  \item defun handleNoParseCommands \quad 717
  \item defun Handle a top level command \quad 718
  \item defun Split block into option block \quad 719
  \item defun Tokenize a system command \quad 719
  \item defun Handle system commands \quad 720
  \item defun Select commands matching this user level \quad 720
  \item defun No command begins with this string \quad 721
  \item defun No option begins with this string \quad 721
  \item defvar $oldline \quad 721
  \item defun No command/option begins with this string \quad 721
  \item defun Option not available at this user level \quad 722
  \item defun Command not available at this user level \quad 722
  \item defun Command not available error message \quad 722
  \item defun satisfiesUserLevel \quad 723
  \item defun hasOption \quad 723
  \item defun terminateSystemCommand \quad 724
  \item defun Terminate a system command \quad 724
  \item defun commandAmbiguityError \quad 724
  \item defun getParserMacroNames \quad 725
  \item defun clearParserMacro \quad 725
  \item defun displayMacro \quad 726
  \item defun displayWorkspaceNames \quad 726
  \item defun getWorkspaceNames \quad 727
  \item defun fixObjectForPrinting \quad 727
  \item defun displayProperties,sayFunctionDeps \quad 728
  \item defun displayValue \quad 731
  \item defun displayType \quad 732
  \item defun getAndSay \quad 732
  \item defun displayProperties \quad 733
  \item defun displayParserMacro \quad 736
  \item defun displayCondition \quad 736
  \item defun interpFunctionDepAlists \quad 737
  \item defun displayModemap \quad 738
  \item defun displayMode \quad 738
  \item defun Split into tokens delimited by spaces \quad 739
  \item defun Convert string tokens to their proper type \quad 739
  \item defun Is the argument string an integer? \quad 740
\end{itemize}
defun Handle parsed system commands ........................................ 740
defun Parse a system command .................................................. 740
defun Get first word in a string ................................................ 741
defun Unabbreviate keywords in commands ................................. 741
defun The command is ambiguous error ........................................ 742
defun Remove the spaces surrounding a string ............................. 743
defun Remove the lisp command prefix ........................................ 743
defun Handle the )lisp command ................................................ 744
defun The )boot command is no longer supported .......................... 744
defun Handle the )system command ............................................. 744
defun Handle the synonym system command .................................. 745
defun printSynonyms ................................................................ 745
defun Print a list of each matching synonym ................................. 746
defvar $tokenCommands ......................................................... 747
defvar $InitialCommandSynonymAlist ......................................... 748
defun Print the current version information .................................. 748
defvar $CommandSynonymAlist .................................................. 749
defun ncloopCommand ................................................................ 750
defun ncloopPrefix? .................................................................. 750
defun selectOptionLC ................................................................ 751
defun selectOption .................................................................... 751
25.3 )abbreviations Command ..................................................... 753
      abbreviations man page ....................................................... 753
defun abbreviations ................................................................. 754
defun abbreviationsSpad2Cmd .................................................... 754
defun listConstructorAbbreviations .......................................... 756
25.4 )boot Command ................................................................ 758
      boot man page .................................................................. 758
25.5 )browse Command ............................................................. 759
      browse man page .............................................................. 759
25.6 Overview .......................................................................... 759
25.7 Browsers, MathML, and Fonts .............................................. 760
25.8 The axServer/multiServ loop ................................................ 761
25.9 The )browse command ....................................................... 761
25.10 The server support code ....................................................... 763
25.11 )cd Command ................................................................ 764
      cd man page .................................................................. 764
25.12 )clear Command .............................................................. 765
      clear man page ............................................................... 765
defvar $clearOptions ................................................................. 766
defun clear ............................................................................. 767
defvar $clearExcept ................................................................. 767
defun clearSpad2Cmd .............................................................. 767
defun clearCmdSortedCaches ..................................................... 769
defun compiledLookupCheck ....................................................... 769
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>xxxiii</th>
</tr>
</thead>
<tbody>
<tr>
<td>defvar $functionTable</td>
<td>770</td>
</tr>
<tr>
<td>defun clearCmdCompletely</td>
<td>770</td>
</tr>
<tr>
<td>defun clearCmdAll</td>
<td>771</td>
</tr>
<tr>
<td>defun clearMacroTable</td>
<td>772</td>
</tr>
<tr>
<td>defun clearCmdExcept</td>
<td>772</td>
</tr>
<tr>
<td>defun clearCmdParts</td>
<td>773</td>
</tr>
<tr>
<td>25.13 )close Command</td>
<td>776</td>
</tr>
<tr>
<td>close man page</td>
<td>776</td>
</tr>
<tr>
<td>defun queryClients</td>
<td>777</td>
</tr>
<tr>
<td>defun close</td>
<td>777</td>
</tr>
<tr>
<td>25.14 )compile Command</td>
<td>779</td>
</tr>
<tr>
<td>compile man page</td>
<td>779</td>
</tr>
<tr>
<td>defvar /editfile</td>
<td>781</td>
</tr>
<tr>
<td>25.15 )copyright Command</td>
<td>782</td>
</tr>
<tr>
<td>copyright man page</td>
<td>782</td>
</tr>
<tr>
<td>defun copyright</td>
<td>787</td>
</tr>
<tr>
<td>defun trademark</td>
<td>787</td>
</tr>
<tr>
<td>25.16 )credits Command</td>
<td>788</td>
</tr>
<tr>
<td>credits man page</td>
<td>788</td>
</tr>
<tr>
<td>defun credits</td>
<td>788</td>
</tr>
<tr>
<td>25.17 )describe Command</td>
<td>789</td>
</tr>
<tr>
<td>describe man page</td>
<td>789</td>
</tr>
<tr>
<td>defvar $describeOptions</td>
<td>789</td>
</tr>
<tr>
<td>defun Print comment strings from algebra libraries</td>
<td>790</td>
</tr>
<tr>
<td>defun describeSpad2Cmd</td>
<td>790</td>
</tr>
<tr>
<td>defun cleanline</td>
<td>791</td>
</tr>
<tr>
<td>defun flatten</td>
<td>793</td>
</tr>
<tr>
<td>25.18 )display Command</td>
<td>794</td>
</tr>
<tr>
<td>display man page</td>
<td>794</td>
</tr>
<tr>
<td>defvar $displayOptions</td>
<td>795</td>
</tr>
<tr>
<td>defun display</td>
<td>796</td>
</tr>
<tr>
<td>displaySpad2Cmd</td>
<td>796</td>
</tr>
<tr>
<td>defun abbQuery</td>
<td>797</td>
</tr>
<tr>
<td>defun displayOperations</td>
<td>798</td>
</tr>
<tr>
<td>defun yesanswer</td>
<td>798</td>
</tr>
<tr>
<td>defun displayMacros</td>
<td>799</td>
</tr>
<tr>
<td>defun sayExample</td>
<td>800</td>
</tr>
<tr>
<td>defun cleanupLine</td>
<td>801</td>
</tr>
<tr>
<td>25.19 )edit Command</td>
<td>803</td>
</tr>
<tr>
<td>edit man page</td>
<td>803</td>
</tr>
<tr>
<td>defun edit</td>
<td>804</td>
</tr>
<tr>
<td>defun editSpad2Cmd</td>
<td>804</td>
</tr>
<tr>
<td>defun Implement the )edit command</td>
<td>805</td>
</tr>
<tr>
<td>defun updateSourceFiles</td>
<td>806</td>
</tr>
<tr>
<td>25.20 )fin Command</td>
<td>808</td>
</tr>
<tr>
<td>fin man page</td>
<td>808</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>25.21 )help Command</td>
<td>810</td>
</tr>
<tr>
<td>help man page</td>
<td>810</td>
</tr>
<tr>
<td>The top level help command</td>
<td>812</td>
</tr>
<tr>
<td>The top level help command handler</td>
<td>813</td>
</tr>
<tr>
<td>defun newHelpSpad2Cmd</td>
<td>813</td>
</tr>
<tr>
<td>25.22 )history Command</td>
<td>815</td>
</tr>
<tr>
<td>history man page</td>
<td>815</td>
</tr>
<tr>
<td>25.23 Initialized history variables</td>
<td>818</td>
</tr>
<tr>
<td>defvar $oldHistoryFileName</td>
<td>818</td>
</tr>
<tr>
<td>defvar $historyFileType</td>
<td>818</td>
</tr>
<tr>
<td>defvar $historyDirectory</td>
<td>819</td>
</tr>
<tr>
<td>defvar $useInternalHistoryTable</td>
<td>819</td>
</tr>
<tr>
<td>defun makeHistFileName</td>
<td>819</td>
</tr>
<tr>
<td>defun oldHistFileName</td>
<td>819</td>
</tr>
<tr>
<td>defun histFileName</td>
<td>820</td>
</tr>
<tr>
<td>defun histInputFileName</td>
<td>820</td>
</tr>
<tr>
<td>defun initHist</td>
<td>820</td>
</tr>
<tr>
<td>defun initHistList</td>
<td>821</td>
</tr>
<tr>
<td>The top level history command</td>
<td>821</td>
</tr>
<tr>
<td>The top level history command handler</td>
<td>822</td>
</tr>
<tr>
<td>defun showHistory</td>
<td>824</td>
</tr>
<tr>
<td>defun setHistoryCore</td>
<td>826</td>
</tr>
<tr>
<td>defvar $underbar</td>
<td>828</td>
</tr>
<tr>
<td>defun writeInputLines</td>
<td>828</td>
</tr>
<tr>
<td>defun resetInCoreHist</td>
<td>830</td>
</tr>
<tr>
<td>defun changeHistListLen</td>
<td>830</td>
</tr>
<tr>
<td>defun updateHist</td>
<td>831</td>
</tr>
<tr>
<td>defun updateInCoreHist</td>
<td>832</td>
</tr>
<tr>
<td>defun putHist</td>
<td>832</td>
</tr>
<tr>
<td>defun recordNewValue</td>
<td>833</td>
</tr>
<tr>
<td>defun recordNewValue0</td>
<td>833</td>
</tr>
<tr>
<td>defun recordOldValue</td>
<td>833</td>
</tr>
<tr>
<td>defun recordOldValue0</td>
<td>834</td>
</tr>
<tr>
<td>defun undoInCore</td>
<td>834</td>
</tr>
<tr>
<td>defun undoChanges</td>
<td>835</td>
</tr>
<tr>
<td>defun undoFromFile</td>
<td>836</td>
</tr>
<tr>
<td>defun saveHistory</td>
<td>837</td>
</tr>
<tr>
<td>defun restoreHistory</td>
<td>839</td>
</tr>
<tr>
<td>defun setIOindex</td>
<td>841</td>
</tr>
<tr>
<td>defun showInput</td>
<td>841</td>
</tr>
<tr>
<td>defun showInOut</td>
<td>842</td>
</tr>
<tr>
<td>defun fetchOutput</td>
<td>842</td>
</tr>
<tr>
<td>Read the history file using index n</td>
<td>843</td>
</tr>
<tr>
<td>Write information of the current step to history file</td>
<td>844</td>
</tr>
<tr>
<td>Disable history if an error occurred</td>
<td>845</td>
</tr>
</tbody>
</table>
defun writeHistModesAndValues ........................................ 845
defun spadrwrite0 .......................................................... 846
defun Random write to a stream .......................................... 846
defun spadrwrite ............................................................. 847
defun spadrread .............................................................. 847
defun Random read a key from a stream .................................. 847
defun unwritable? .................................................................. 848
defun writifyComplain ......................................................... 848
defun safeWritify ............................................................... 849
defun writify,writifyInner ..................................................... 849
defun writify ................................................................. 852
defun spadClosure? ............................................................. 853
defvar $NonNullStream ......................................................... 853
defvar $NullStream ............................................................ 854
defun dewritify,dewritifyInner ............................................... 854
defun dewritify ............................................................... 857
defun ScanOrPairVec,ScanOrInner ........................................... 857
defun ScanOrPairVec .......................................................... 858
defun gensymInt .............................................................. 859
defun charDigitVal ............................................................ 859
defun histFileErase ............................................................ 860
25.24 )include Command ......................................................... 861
   include man page ........................................................... 861
   defun ncloopInclude1 ....................................................... 861
   Returns the first non-blank substring of the given string ......... 861
   Open the include file and read it in .................................. 862
   Return the include filename ............................................. 862
   Return the next token ..................................................... 862
25.25 )library Command ......................................................... 864
   library man page .......................................................... 864
25.26 )license Command ........................................................ 866
   license man page ........................................................ 866
   defun license .............................................................. 866
25.27 )lisp Command ............................................................ 867
   lisp man page ............................................................ 867
25.28 )ltrace Command .......................................................... 868
   ltrace man page .......................................................... 868
   defun The top level )ltrace function .................................. 868
25.29 )pquit Command .......................................................... 869
   pquit man page .......................................................... 869
   The top level pquit command .......................................... 870
   The top level pquit command handler ................................... 870
25.30 )quit Command ............................................................ 871
   quit man page ............................................................ 871
   The top level quit command .......................................... 872
   The top level quit command handler ................................... 872
### Leave the Axiom interpreter
- 25.31 `read` Command
  - `read` man page
  - `defun The `read` command`
  - `defun Implement the `read` command`
  - `defun /read`
- 25.32 `regress` Command
  - `regress` man page
  - `defvar *all-tests-ran*`
  - `defun Scan a spool output file for failures`
  - `defun Parse test name from the spool command`
  - `defun Find the next –S marker`
  - `defun Parse out the test number from –S lines`
  - `defvar *ok*`
  - `defun Compare the computed and expected results`
  - `defun Split the calculated and expect results into lists`
  - `defun Returns true on –S lines`
  - `defun Returns true on –E lines`
  - `defun Returns true on –R lines`
  - `defun Returns true on –I lines`
  - `defun Check the last –S line ran`
- 25.33 `savesystem` Command
  - `savesystem` man page
  - `defvar *ThisIsARunningSystem*`
  - `defun The `savesystem` command`
- 25.34 `set` Command
  - `set` man page
  - `defvar $BreakMode`
  - `defvar $lambdatype`
  - `set compiler output`
  - `The set output command handler`
  - `Describe the set output library arguments`
  - `defvar output-library`
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>25.38</td>
<td>set debug dalymode</td>
<td>908</td>
</tr>
<tr>
<td></td>
<td>defvar dalymode</td>
<td>908</td>
</tr>
<tr>
<td>25.39</td>
<td>set expose</td>
<td>909</td>
</tr>
<tr>
<td></td>
<td>functions</td>
<td>910</td>
</tr>
<tr>
<td></td>
<td>functions cache</td>
<td>910</td>
</tr>
<tr>
<td></td>
<td>defvar $cacheAlist</td>
<td>911</td>
</tr>
<tr>
<td></td>
<td>The top level set functions cache handler</td>
<td>911</td>
</tr>
<tr>
<td></td>
<td>Display a particular cache count</td>
<td>912</td>
</tr>
<tr>
<td></td>
<td>defun insertAlist</td>
<td>913</td>
</tr>
<tr>
<td></td>
<td>Describe the set functions cache</td>
<td>914</td>
</tr>
<tr>
<td></td>
<td>Display all cache counts</td>
<td>914</td>
</tr>
<tr>
<td></td>
<td>Describe the cache counts</td>
<td>915</td>
</tr>
<tr>
<td></td>
<td>functions compile</td>
<td>916</td>
</tr>
<tr>
<td></td>
<td>defvar $compileRecurrence</td>
<td>917</td>
</tr>
<tr>
<td>25.40</td>
<td>set fortran</td>
<td>917</td>
</tr>
<tr>
<td></td>
<td>set ints2floats</td>
<td>918</td>
</tr>
<tr>
<td></td>
<td>defvar $fortInts2Floats</td>
<td>918</td>
</tr>
<tr>
<td></td>
<td>set fortindent</td>
<td>919</td>
</tr>
<tr>
<td></td>
<td>defvar $fortIndent</td>
<td>919</td>
</tr>
<tr>
<td></td>
<td>set fortlength</td>
<td>920</td>
</tr>
<tr>
<td></td>
<td>defvar $fortLength</td>
<td>920</td>
</tr>
<tr>
<td></td>
<td>set typedecs</td>
<td>920</td>
</tr>
<tr>
<td></td>
<td>defvar $printFortranDecs</td>
<td>921</td>
</tr>
<tr>
<td></td>
<td>set defaulttype</td>
<td>921</td>
</tr>
<tr>
<td></td>
<td>defvar $defaultFortranType</td>
<td>921</td>
</tr>
<tr>
<td></td>
<td>set precision</td>
<td>922</td>
</tr>
<tr>
<td></td>
<td>defvar $fortranPrecision</td>
<td>922</td>
</tr>
<tr>
<td></td>
<td>set intrinsic</td>
<td>923</td>
</tr>
<tr>
<td></td>
<td>defvar $useIntrinsicFunctions</td>
<td>923</td>
</tr>
<tr>
<td></td>
<td>set explength</td>
<td>923</td>
</tr>
<tr>
<td></td>
<td>defvar $maximumFortranExpressionLength</td>
<td>924</td>
</tr>
<tr>
<td></td>
<td>set segment</td>
<td>924</td>
</tr>
<tr>
<td></td>
<td>defvar $fortranSegment</td>
<td>925</td>
</tr>
<tr>
<td></td>
<td>set optlevel</td>
<td>925</td>
</tr>
<tr>
<td></td>
<td>defvar $fortranOptimizationLevel</td>
<td>925</td>
</tr>
<tr>
<td></td>
<td>set startindex</td>
<td>926</td>
</tr>
<tr>
<td></td>
<td>defvar $fortranArrayStartingIndex</td>
<td>926</td>
</tr>
<tr>
<td></td>
<td>set calling</td>
<td>926</td>
</tr>
<tr>
<td></td>
<td>defvar $fortranTmpDir</td>
<td>927</td>
</tr>
</tbody>
</table>
The top level set fortran calling temp file handler .......................... 928
Validate the output directory ....................................................... 929
Describe the set fortran calling temp file ..................................... 929
defvar $fortranDirectory .......................................................... 930
defun setFortDir ................................................................. 930
defun describeSetFortDir ......................................................... 931
defvar $fortranLibraries .......................................................... 932
defun setLinkerArgs .................................................................. 933
defun describeSetLinkerArgs ....................................................... 933
25.41 set hyperdoc ................................................................. 934
call screen ............................................................................... 934
defvar $fullScreenSysVars ......................................................... 934
mathwidth .............................................................................. 935
defvar $historyDisplayWidth ..................................................... 935
25.42 set help ........................................................................... 936
call screen .............................................................................. 936
defvar $useFullScreenHelp ......................................................... 936
25.43 set history ....................................................................... 937
defvar $HiFiAccess ................................................................. 937
25.44 set messages ................................................................. 938
set message any ...................................................................... 939
defvar $printAnyIfTrue ............................................................ 939
set message autoload .................................................................. 940
defvar $printLoadMsgs .............................................................. 940
set message bottomup ................................................................ 940
defvar $reportBottomUpFlag ...................................................... 941
set message coercion ................................................................. 941
defvar $reportCoerceIfTrue ......................................................... 941
set message dropmap .................................................................. 942
defvar $displayDroppedMap ....................................................... 942
set message expose ................................................................... 943
defvar $giveExposureWarning .................................................... 943
set message file ....................................................................... 944
defvar $printMsgsToFile ........................................................... 944
set message frame .................................................................... 945
defvar $frameMessages ............................................................. 945
set message highlighting ............................................................. 945
defvar $highlightAllowed .......................................................... 945
set message instant ................................................................... 946
defvar $reportInstantiations ...................................................... 946
set message insteach .................................................................. 947
defvar $reportEachInstantiation ................................................ 947
set message interponly ............................................................... 948
defvar $reportInterpOnly .......................................................... 948
set message naglink ................................................................... 949
defvar $nagMessages ............................................................... 949
defun describeSetOutputFortran ........................................... 977
set output fraction .................................................. 977
defvar $fractionDisplayType ............................................. 978
set output html ...................................................... 978
defvar $htmlFormat ................................................... 979
defvar $htmlOutputFile .................................................. 979
defun setOutputHtml .................................................. 980
defun describeSetOutputHtml .......................................... 982
set output length ................................................... 983
defvar $margin ....................................................... 983
defvar $linelength .................................................. 983
set output mathml .................................................... 984
defvar $mathmlFormat ................................................ 985
defvar $mathmlOutputFile ............................................. 985
defun setOutputMathml ................................................ 986
defun describeSetOutputMathml .................................... 988
set output openmath ................................................. 989
defvar $openMathFormat .............................................. 989
defvar $openMathOutputFile .......................................... 990
defun setOutputOpenMath ............................................. 990
defun describeSetOutputOpenMath .................................. 993
set output script ................................................... 993
defvar $formulaFormat ............................................... 994
defvar $formulaOutputFile ........................................... 994
defun setOutputFormula ............................................... 995
defun describeSetOutputFormula ..................................... 997
set output scripts ................................................ 998
defvar $linearFormatScripts ........................................ 999
set output showeditor ............................................... 999
defvar $useEditorForShowOutput .................................... 999
set output tex ...................................................... 1000
defvar $texFormat .................................................. 1001
defvar $texOutputFile ............................................... 1001
defun setOutputTex .................................................. 1001
defun describeSetOutputTex ......................................... 1004
25.47 quit .......................................................... 1005
defvar $quitCommandType ............................................. 1005
25.48 streams ....................................................... 1005
set streams calculate .............................................. 1006
defvar $streamCount ................................................ 1006
defun setStreamsCalculate .......................................... 1007
defun describeSetStreamsCalculate ............................... 1007
set streams showall ................................................ 1008
defvar $streamsShowAll ............................................. 1008
25.49 set system ..................................................... 1009
set system functioncode ............................................. 1009
<table>
<thead>
<tr>
<th>Function/Variable/Procedure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>25.59 )trace Command</td>
<td>1045</td>
</tr>
<tr>
<td>trace man page</td>
<td>1045</td>
</tr>
<tr>
<td>The trace global variables</td>
<td>1049</td>
</tr>
<tr>
<td>defvar $traceNoisely</td>
<td>1049</td>
</tr>
<tr>
<td>defvar $reportSpadtrace</td>
<td>1050</td>
</tr>
<tr>
<td>defvar $optionAlist</td>
<td>1050</td>
</tr>
<tr>
<td>defvar $tracedMapSignatures</td>
<td>1050</td>
</tr>
<tr>
<td>defvar $traceOptionList</td>
<td>1050</td>
</tr>
<tr>
<td>defun resetTimers</td>
<td>1051</td>
</tr>
<tr>
<td>defun resetSpacers</td>
<td>1051</td>
</tr>
<tr>
<td>defun resetCounters</td>
<td>1051</td>
</tr>
<tr>
<td>defun ptimers</td>
<td>1052</td>
</tr>
<tr>
<td>defun pspacers</td>
<td>1052</td>
</tr>
<tr>
<td>defun pcounters</td>
<td>1053</td>
</tr>
<tr>
<td>defun transOnlyOption</td>
<td>1053</td>
</tr>
<tr>
<td>defun stackTraceOptionError</td>
<td>1054</td>
</tr>
<tr>
<td>defun removeOption</td>
<td>1054</td>
</tr>
<tr>
<td>defun domainToGenvar</td>
<td>1054</td>
</tr>
<tr>
<td>defun subTypes</td>
<td>1055</td>
</tr>
<tr>
<td>defun isListOfIdentifiers</td>
<td>1055</td>
</tr>
<tr>
<td>defun isListOfIdentifiersOrStrings</td>
<td>1056</td>
</tr>
<tr>
<td>defun getPreviousMapSubNames</td>
<td>1056</td>
</tr>
<tr>
<td>defun lassocSub</td>
<td>1057</td>
</tr>
<tr>
<td>defun rassocSub</td>
<td>1058</td>
</tr>
<tr>
<td>defun isUncompiledMap</td>
<td>1058</td>
</tr>
<tr>
<td>defun isInterpOnlyMap</td>
<td>1058</td>
</tr>
<tr>
<td>defun isSubForRedundantMapName</td>
<td>1059</td>
</tr>
<tr>
<td>defun untraceMapSubNames</td>
<td>1059</td>
</tr>
<tr>
<td>defun funfind,LAM</td>
<td>1060</td>
</tr>
<tr>
<td>defmacro funfind</td>
<td>1060</td>
</tr>
<tr>
<td>defun isDomainOrPackage</td>
<td>1061</td>
</tr>
<tr>
<td>defun isTraceGensym</td>
<td>1061</td>
</tr>
<tr>
<td>defun flattenOperationAlist</td>
<td>1061</td>
</tr>
<tr>
<td>defun letPrint</td>
<td>1062</td>
</tr>
<tr>
<td>defun Identifier beginning with a sharpsign-number?</td>
<td>1063</td>
</tr>
<tr>
<td>defun Identifier beginning with a sharpsign?</td>
<td>1064</td>
</tr>
<tr>
<td>defun isgenvar</td>
<td>1064</td>
</tr>
<tr>
<td>defun letPrint2</td>
<td>1064</td>
</tr>
<tr>
<td>defun letPrint3</td>
<td>1066</td>
</tr>
<tr>
<td>defun hasPair</td>
<td>1067</td>
</tr>
<tr>
<td>defun shortenForPrinting</td>
<td>1067</td>
</tr>
<tr>
<td>defun getOption</td>
<td>1068</td>
</tr>
<tr>
<td>defun orderBySlotNumber</td>
<td>1068</td>
</tr>
<tr>
<td>defun spadReply,printName</td>
<td>1069</td>
</tr>
<tr>
<td>defun spadReply</td>
<td>1069</td>
</tr>
<tr>
<td>defun remover</td>
<td>1070</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>25.60    undo Command</td>
<td>1072</td>
</tr>
<tr>
<td>25.61    Evaluation</td>
<td>1073</td>
</tr>
<tr>
<td>25.62    what Command</td>
<td>1089</td>
</tr>
<tr>
<td>25.63    workfiles Command</td>
<td>1099</td>
</tr>
<tr>
<td>26.6     Handlers for Special Forms</td>
<td>1103</td>
</tr>
</tbody>
</table>

defvar $constructors ........................................... 1070
defun stupidIsSpadFunction ............................... 1070
defun compileBoot ........................................... 1071

undo man page .................................................. 1072

evalDomain ....................................................... 1075
defun mkEvalable ............................................... 1075
defun mkEvalableUnion ......................................... 1077
defun isTaggedUnion ........................................... 1077
defun mkEvalableRecord ........................................ 1078
defun mkEvalableMapping ....................................... 1078
defun evaluateType ............................................. 1078
defun Eval args passed to a constructor .................. 1080
defvar $noEvalTypeMsg ......................................... 1082
defun throwEvalTypeMsg ......................................... 1082
defun makeOrdinal ............................................... 1083
defun evaluateSignature ........................................ 1083
defun recordFrame ............................................... 1083
defun diffAlist .................................................. 1085
defun clearFrame ................................................. 1088

what man page .................................................... 1089
defvar $whatOptions ............................................ 1090
defun what .......................................................... 1091
defun whatSpad2Cmd, xpath .................................... 1091
defun whatSpad2Cmd ............................................... 1091
defun Show keywords for )what command ..................... 1093
defun The )what commands implementation .................... 1093
defun Find all names contained in a pattern .............. 1094
defun Find function of names contained in pattern ........ 1095
defun satisfiesRegularExpressions ......................... 1095
defun filterAndFormatConstructors ............................ 1096
defun whatConstructors .......................................... 1097

Display all operation names containing the fragment ....... 1097

workfiles man page ............................................... 1099
defun workfiles ................................................... 1099
defun workfilesSpad2Cmd ......................................... 1099

getAndEvalConstructorArgument .................................. 1104
defun replaceSharps .............................................. 1104
defun isDomainValuedVariable ................................... 1105
defun evalCategory ............................................... 1105
27 Handling input files

- defun Handle .axiom.input file ........................................ 1107
- defvar boot-line-stack .................................................. 1107
- defvar in-stream .......................................................... 1107
- defvar out-stream .......................................................... 1108
- defvar file-closed .......................................................... 1108
- defvar echo-meta .......................................................... 1108
- defvar $noSubsumption .................................................... 1108
- defvar $envHashTable ...................................................... 1109
- defun Dynamically add bindings to the environment .............. 1109
- defun Fetch a property list for a symbol from CategoryFrame ... 1110
- defun Search for a binding in the environment list ............... 1110
- defun Search for a binding in the current environment .......... 1110
- defun searchTailEnv ....................................................... 1111

28 File Parsing

- defun Bind a variable in the interactive environment .......... 1113
- defvar line-handler ....................................................... 1113
- defvar $spad-errors ...................................................... 1113
- defvar xtokenreader ...................................................... 1114
- defun Initialize the spad reader ...................................... 1114
- defun spad-syntax-error ................................................ 1114
- defun spad-long-error ................................................... 1115
- defun spad-short-error .................................................. 1115
- defun spad-error-loc ..................................................... 1116
- defun iostat ............................................................... 1116
- defun next-lines-show ................................................... 1117
- defun token-stack-show ................................................ 1117
- defun ioclear ............................................................. 1118
- defun Set boot-line-stack to nil ...................................... 1118

29 Handling output

29.1 Special Character Tables ............................................. 1121
- defvar $defaultSpecialCharacters .................................... 1121
- defvar $plainSpecialCharacters0 ..................................... 1122
- defvar $plainSpecialCharacters1 ..................................... 1122
- defvar $plainSpecialCharacters2 ..................................... 1123
- defvar $plainSpecialCharacters3 ..................................... 1123
- defvar $plainRTspecialCharacters ................................... 1124
- defvar $RTspecialCharacters .......................................... 1124
- defvar $specialCharacters ............................................. 1125
- defvar $specialCharacterAlist ....................................... 1125
- defun Look up a special character code for a symbol .......... 1126
30 Stream and File Handling

```lisp
(defun make-instream ................. 1127
(defun make-outstream ............... 1127
(defun make-appendstream .......... 1128
(defun defiostream .................. 1128
(defun shut ......................... 1128
(defun eofp ........................ 1129
(defun makeStream ................. 1129
(defun Construct a new input file name 
   ....................... 1129
(defun getDirectoryList .......... 1130
(defun probeName ................... 1130
(defun makeFullNamestring ...... 1131
(defun Replace a file by erase and rename .
   ....................... 1131
```

31 The Spad Server Mechanism

```lisp
(defun openserver .................. 1133
```

32 Axiom Build-time Functions

```lisp
(defun spad-save ...................... 1135
```

33 Exposure Groups

34 Databases

34.1 Database structure .............. 1139

```lisp
(kaf File Format .................. 1139
(Database Files ................... 1140
(defstruct $database ............... 1142
(defvar *defaultdomain-list* .... 1142
(defvar *operation-hash* ......... 1143
(defvar *hasCategory-hash* ....... 1143
(defvar *miss* .................... 1143
(Database streams ................. 1144
(defvar *interp-stream* ........... 1144
(defvar *interp-stream-stamp* .... 1144
(defvar *operation-stream* ....... 1144
(defvar *operation-stream-stamp* . 1145
(defvar *browse-stream* .......... 1145
(defvar *browse-stream-stamp* ..... 1145
(defvar *category-stream* ......... 1145
(defvar *category-stream-stamp* ... 1145
(defvar *allconstructors* ......... 1146
(defvar *allOperations* .......... 1146
(defun Reset all hash tables before saving system 
   ....................... 1146
(defun Preload algebra into saved system .... 1147
(defun Open the interp database ........ 1149
(defun Open the browse database .... 1151
```
defun Open the category database ........................................... 1152
defun Open the operations database ........................................ 1153
defun Add operations from newly compiled code .......................... 1153
defun Show all database attributes of a constructor .................... 1154
defun Set a value for a constructor key in the database ................. 1155
defun Delete a value for a constructor key in the database ............. 1155
defun Get constructor information for a database key ................... 1156
defun The )library top level command ..................................... 1159
defun Read a local filename and update the hash tables ................. 1160
defun Update the database from an nrlib index.kaf file ................ 1161
defun updateDatabase .......................................................... 1163
defvar *sourcefiles* .......................................................... 1164
defun Make new databases ..................................................... 1164
defun saveDependentsHashTable .............................................. 1168
defun saveUsersHashTable .................................................... 1168
defun Construct the proper database full pathname ......................... 1169
Building the interp database from hash tables ............................ 1169
defun Write the interp database ............................................ 1173
Building the browse database from hash tables ............................ 1175
defun Write the browse database ............................................ 1175
Building the category database from hash tables ........................ 1176
defun Write the category database .......................................... 1176
Building the operation database from hash tables ......................... 1177
defun Write the operations database ....................................... 1177
Database support operations .................................................. 1178
defun Data preloaded into the image at build time ....................... 1178
defun Return all constructors ................................................ 1178
defun Return all operations .................................................. 1179

35 System Statistics ............................ 1181
   defun statisticsInitialization ............................................ 1181
35.1 Lisp Library Handling .................................................. 1181
defun loadLib .............................................................. 1181
defun isSystemDirectory .................................................... 1183
defun loadLibNoUpdate ...................................................... 1183
defun loadFunctor ........................................................... 1184

36 Special Lisp Functions .................................................. 1185
   defun compiledLookup ..................................................... 1185
defmacro hashCode? .......................................................... 1185
defun basicLookup .......................................................... 1185
defun lookupInDomainVector ................................................. 1187
defun basicLookupCheckDefaults .......................................... 1188
defun oldCompLookup ........................................................ 1188
defun NRTevalDomain ......................................................... 1189
36.1 Axiom control structure macros ....................................... 1189
### 36.2 Filename Handling
- defun namestring .................................................. 1190
- defun pathnameName .............................................. 1191
- defun pathnameType .............................................. 1191
- defun pathnameTypeId ........................................... 1191
- defun pathnameDirectory ........................................ 1192
- defun Axiom pathnames ......................................... 1192
- defun makePathname ............................................. 1193
- defun Delete a File .............................................. 1193
- defun wrap ....................................................... 1193
- defun lotsof ...................................................... 1194
- defmacro startsId? ................................................ 1194
- defun hput ........................................................ 1194
- defmacro hget ...................................................... 1194
- defun hkeys ....................................................... 1195
- defun digitp ........................................................ 1195
- defun pname ........................................................ 1195
- defun size .......................................................... 1196
- defun strpos ........................................................ 1196
- defun strposl ....................................................... 1196
- defmacro identp .................................................... 1197
- defun concat ....................................................... 1197
- defun canFuncall? ................................................ 1197
- defun brightprint ................................................ 1198
- defun brightprint-0 .............................................. 1198
- defun messageprint .............................................. 1199
- defun messageprint-1 ............................................ 1199
- defun messageprint-2 ............................................ 1199
- defun sayBrightly1 ............................................... 1200
- defmacro assq ....................................................... 1200
- defun A version of GET that works with lists ............... 1200

### 37 Record, Union, Mapping, and Enumeration

### 38 Common Lisp Algebra Support

#### 38.1 AlgebraicFunction
- defun retract ....................................................... 1203

#### 38.2 Any
- defun spad2BootCoerce ......................................... 1205

#### 38.3 ApplicationProgramInterface
- defun Report what domains get instantiated ............... 1205
38.4 Boolean ......................................................... 1206
    defun The Boolean = function support .............. 1206
38.5 Char .......................................................... 1206
    defun upcase ............................................. 1206
    defun downcase ......................................... 1206
38.6 ComplexDoubleFloatMatrix .............................. 1207
    defmacro make-cdouble-matrix ......................... 1207
    defmacro cdaref2 ........................................ 1207
    defmacro cdsetaref2 .................................... 1207
    defmacro cdanrows ...................................... 1207
    defmacro cdancols ...................................... 1208
38.7 ComplexDoubleFloatVector .............................. 1208
    defmacro make-cdouble-vector ......................... 1208
    defmacro cdelt ........................................... 1209
    defmacro cdsetelt ....................................... 1209
    defmacro cdlen ......................................... 1210
38.8 Database ..................................................... 1210
    defun Database elt function support ............... 1210
38.9 DirectProduct .............................................. 1210
    defun vec2list .......................................... 1210
38.10 DoubleFloat ................................................. 1210
    defmacro DFLessThan .................................... 1211
    defmacro DFUnaryMinus ................................ 1211
    defmacro DFMinusp ...................................... 1211
    defmacro DFZerop ....................................... 1211
    defmacro DFAdd .......................................... 1212
    defmacro DFSubtract ................................... 1212
    defmacro DFMultiply .................................... 1212
    defmacro DFIntegerMultiply ............................ 1213
    defmacro DFMax .......................................... 1213
    defmacro DFMin .......................................... 1213
    defmacro DFEql .......................................... 1213
    defmacro DFDivide ...................................... 1213
    defmacro DFIntegerDivide ............................... 1214
    defmacro DFSqrt ......................................... 1214
    defmacro DFLogE .......................................... 1214
    defmacro DFLog .......................................... 1214
    defmacro DFIntegerExpt ................................. 1215
    defmacro DFExpt ......................................... 1215
    defmacro DFExp .......................................... 1216
    defmacro DFSin .......................................... 1216
    defmacro DFCos .......................................... 1216
    defmacro DFTan .......................................... 1217
### CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>defmacro DFAtan2</td>
<td>1217</td>
</tr>
<tr>
<td>defmacro DFSinh</td>
<td>1217</td>
</tr>
<tr>
<td>defmacro DFCosh</td>
<td>1218</td>
</tr>
<tr>
<td>defmacro DFTanh</td>
<td>1218</td>
</tr>
<tr>
<td>defmacro DFAsinh</td>
<td>1218</td>
</tr>
<tr>
<td>defmacro DFAcosh</td>
<td>1219</td>
</tr>
<tr>
<td>defmacro DFAtanh</td>
<td>1219</td>
</tr>
<tr>
<td>defun Machine specific float numerator</td>
<td>1219</td>
</tr>
<tr>
<td>defun Machine specific float denominator</td>
<td>1220</td>
</tr>
<tr>
<td>defun Machine specific float sign</td>
<td>1220</td>
</tr>
<tr>
<td>defun Machine specific float bit length</td>
<td>1220</td>
</tr>
<tr>
<td>defun Decode floating-point values</td>
<td>1220</td>
</tr>
<tr>
<td>defun The cotangent routine</td>
<td>1221</td>
</tr>
<tr>
<td>defun The inverse cotangent function</td>
<td>1221</td>
</tr>
<tr>
<td>defun The secant function</td>
<td>1221</td>
</tr>
<tr>
<td>defun The inverse secant function</td>
<td>1222</td>
</tr>
<tr>
<td>defun The cosecant function</td>
<td>1222</td>
</tr>
<tr>
<td>defun The inverse cosecant function</td>
<td>1222</td>
</tr>
<tr>
<td>defun The hyperbolic cosecant function</td>
<td>1223</td>
</tr>
<tr>
<td>defun The hyperbolic cotangent function</td>
<td>1223</td>
</tr>
<tr>
<td>defun The hyperbolic secant function</td>
<td>1223</td>
</tr>
<tr>
<td>defun The inverse hyperbolic cosecant function</td>
<td>1223</td>
</tr>
<tr>
<td>defun The inverse hyperbolic cotangent function</td>
<td>1224</td>
</tr>
<tr>
<td>defun The inverse hyperbolic secant function</td>
<td>1224</td>
</tr>
</tbody>
</table>

#### 38.11 DoubleFloatMatrix
- defmacro make-double-matrix | 1224
- defmacro make-double-matrix1 | 1224
- defmacro daref2 | 1225
- defmacro dsetaref2 | 1225
- defmacro danrows | 1225
- defmacro dancols | 1226

#### 38.12 DoubleFloatVector
- defmacro dlen | 1226
- defmacro make-double-vector | 1226
- defmacro make-double-vector1 | 1226
- defmacro delt | 1227
- defmacro dsetelt | 1227

#### 38.13 File
- defvar *read-place-holder* | 1227
- defun placep | 1227
- defun vmread | 1228

#### 38.14 FileName
- defun FileName filename function implementation | 1228
- defun FileName filename support function | 1228
- defun FileName directory function implementation | 1229
- defun FileName directory function support | 1229
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>38.15 IndexedBits</td>
<td>defun IndexedBits new function support</td>
</tr>
<tr>
<td></td>
<td>defmacro truth-to-bit</td>
</tr>
<tr>
<td></td>
<td>defmacro bvec-elt</td>
</tr>
<tr>
<td></td>
<td>defmacro bvec-setelt</td>
</tr>
<tr>
<td></td>
<td>defmacro bvec-size</td>
</tr>
<tr>
<td></td>
<td>defun IndexedBits concat function support</td>
</tr>
<tr>
<td></td>
<td>defun IndexedBits copy function support</td>
</tr>
<tr>
<td></td>
<td>defun IndexedBits = function support</td>
</tr>
<tr>
<td></td>
<td>defun IndexedBits &lt; function support</td>
</tr>
<tr>
<td></td>
<td>defun IndexedBits And function support</td>
</tr>
<tr>
<td></td>
<td>defun IndexedBits Or function support</td>
</tr>
<tr>
<td></td>
<td>defun IndexedBits xor function support</td>
</tr>
<tr>
<td></td>
<td>defun IndexedBits nand function support</td>
</tr>
<tr>
<td></td>
<td>defun IndexedBits nor function support</td>
</tr>
<tr>
<td></td>
<td>defun IndexedBits not function support</td>
</tr>
<tr>
<td>38.16 IndexCard</td>
<td>defun IndexCard origin function support</td>
</tr>
<tr>
<td></td>
<td>defun IndexCard origin function support</td>
</tr>
<tr>
<td></td>
<td>defun IndexCard elt function support</td>
</tr>
<tr>
<td>38.17 IndexedString</td>
<td>defun qenum</td>
</tr>
<tr>
<td>38.18 InputForm</td>
<td>defun called by interpret function</td>
</tr>
<tr>
<td></td>
<td>defun unparseInputForm</td>
</tr>
<tr>
<td>38.19 Integer</td>
<td>defun Integer divide function support</td>
</tr>
<tr>
<td></td>
<td>defun Integer quo function support</td>
</tr>
<tr>
<td></td>
<td>defun Integer random function support</td>
</tr>
<tr>
<td>38.20 KeyedAccessFile</td>
<td>defun KeyedAccessFile defstream function support</td>
</tr>
<tr>
<td>38.21 NumberFormats</td>
<td>defun ncParseFromString</td>
</tr>
<tr>
<td>38.22 OperationsQuery</td>
<td></td>
</tr>
<tr>
<td>Section</td>
<td>Topic</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td></td>
<td>defun OperationQuery getDatabase</td>
</tr>
<tr>
<td></td>
<td>function support</td>
</tr>
<tr>
<td>38.23</td>
<td>ParametricLinearEquations</td>
</tr>
<tr>
<td></td>
<td>defun algCoerceInteractive</td>
</tr>
<tr>
<td>38.24</td>
<td>Plot3d</td>
</tr>
<tr>
<td></td>
<td>defvar $numericFailure</td>
</tr>
<tr>
<td></td>
<td>defvar $oldBreakMode</td>
</tr>
<tr>
<td></td>
<td>defmacro trapNumericErrors</td>
</tr>
<tr>
<td>38.25</td>
<td>SingleInteger</td>
</tr>
<tr>
<td></td>
<td>defun qsquotient</td>
</tr>
<tr>
<td></td>
<td>defun qremainder</td>
</tr>
<tr>
<td></td>
<td>defmacro qsdifference</td>
</tr>
<tr>
<td></td>
<td>defmacro qslessp</td>
</tr>
<tr>
<td></td>
<td>defmacro qsdadd1</td>
</tr>
<tr>
<td></td>
<td>defmacro qssub1</td>
</tr>
<tr>
<td></td>
<td>defmacro qsminus</td>
</tr>
<tr>
<td></td>
<td>defmacro qsplus</td>
</tr>
<tr>
<td></td>
<td>defmacro qstimes</td>
</tr>
<tr>
<td></td>
<td>defmacro qsabsval</td>
</tr>
<tr>
<td></td>
<td>defmacro qsdaddp</td>
</tr>
<tr>
<td></td>
<td>defmacro qszerop</td>
</tr>
<tr>
<td></td>
<td>defmacro qsmax</td>
</tr>
<tr>
<td></td>
<td>defmacro qsmint</td>
</tr>
<tr>
<td>38.26</td>
<td>Table</td>
</tr>
<tr>
<td></td>
<td>defun Table InnerTable</td>
</tr>
<tr>
<td>38.27</td>
<td>U8Vector</td>
</tr>
<tr>
<td></td>
<td>defmacro qvlenU8</td>
</tr>
<tr>
<td></td>
<td>defmacro eltU8</td>
</tr>
<tr>
<td></td>
<td>defmacro seteltU8</td>
</tr>
<tr>
<td></td>
<td>defun getRefvU8</td>
</tr>
<tr>
<td>38.28</td>
<td>U16Vector</td>
</tr>
<tr>
<td></td>
<td>defmacro qvlenU16</td>
</tr>
<tr>
<td></td>
<td>defmacro eltU16</td>
</tr>
<tr>
<td></td>
<td>defmacro seteltU16</td>
</tr>
<tr>
<td></td>
<td>defun getRefvU16</td>
</tr>
<tr>
<td>38.29</td>
<td>U32Vector</td>
</tr>
<tr>
<td></td>
<td>defmacro qvlenU32</td>
</tr>
<tr>
<td></td>
<td>defmacro eltU32</td>
</tr>
<tr>
<td></td>
<td>defmacro seteltU32</td>
</tr>
<tr>
<td></td>
<td>defun getRefvU32</td>
</tr>
<tr>
<td>38.30</td>
<td>U8Matrix</td>
</tr>
<tr>
<td></td>
<td>defmacro aref2U8</td>
</tr>
<tr>
<td></td>
<td>defmacro setAref2U8</td>
</tr>
<tr>
<td></td>
<td>defmacro anrowsU8</td>
</tr>
<tr>
<td></td>
<td>defmacro ancolsU8</td>
</tr>
<tr>
<td></td>
<td>defmacro makeMatrixU8</td>
</tr>
<tr>
<td></td>
<td>defmacro makeMatrix1U8</td>
</tr>
</tbody>
</table>
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>39.9 Comments in the SGML/XML Encodings</td>
<td>1274</td>
</tr>
<tr>
<td>39.10 I/O Functions for Devices</td>
<td>1275</td>
</tr>
<tr>
<td>39.11 Communications</td>
<td>1275</td>
</tr>
<tr>
<td>Functions to Initiate an <code>OMconn</code></td>
<td>1276</td>
</tr>
<tr>
<td>39.12 Parameters</td>
<td>1277</td>
</tr>
<tr>
<td>39.13 Miscellaneous Functions and Variables</td>
<td>1277</td>
</tr>
<tr>
<td>39.14 The <code>OM.h</code> header file</td>
<td>1278</td>
</tr>
<tr>
<td>39.15 Axiom OpenMath stub functions</td>
<td>1287</td>
</tr>
<tr>
<td>Axiom specific functions</td>
<td>1287</td>
</tr>
<tr>
<td><code>defun om-Read</code></td>
<td>1287</td>
</tr>
<tr>
<td><code>defun om-listCDs</code></td>
<td>1288</td>
</tr>
<tr>
<td><code>defun om-listSymbols</code></td>
<td>1288</td>
</tr>
<tr>
<td><code>defun om-supportsCD</code></td>
<td>1288</td>
</tr>
<tr>
<td><code>defun om-supportsSymbol</code></td>
<td>1288</td>
</tr>
<tr>
<td>Lisp conversion functions</td>
<td>1289</td>
</tr>
<tr>
<td><code>defun om-setDevEncoding</code></td>
<td>1289</td>
</tr>
<tr>
<td>Device manipulation functions</td>
<td>1289</td>
</tr>
<tr>
<td><code>defun om-openFileDev</code></td>
<td>1289</td>
</tr>
<tr>
<td><code>defun om-openStringDev</code></td>
<td>1290</td>
</tr>
<tr>
<td><code>defun om-closeDev</code></td>
<td>1290</td>
</tr>
<tr>
<td>Connection manipulation functions</td>
<td>1290</td>
</tr>
<tr>
<td><code>defun om-makeConn</code></td>
<td>1290</td>
</tr>
<tr>
<td><code>defun om-closeConn</code></td>
<td>1290</td>
</tr>
<tr>
<td><code>defun om-getConnInDev</code></td>
<td>1291</td>
</tr>
<tr>
<td><code>defun om-getConnOutDev</code></td>
<td>1291</td>
</tr>
<tr>
<td>Client/Server functions</td>
<td>1291</td>
</tr>
<tr>
<td><code>defun om-bindTCP</code></td>
<td>1291</td>
</tr>
<tr>
<td><code>defun om-connectTCP</code></td>
<td>1292</td>
</tr>
<tr>
<td>Device input/output functions</td>
<td>1292</td>
</tr>
<tr>
<td><code>defun om-getApp</code></td>
<td>1293</td>
</tr>
<tr>
<td><code>defun om-getAtp</code></td>
<td>1294</td>
</tr>
<tr>
<td><code>defun om-getAttr</code></td>
<td>1294</td>
</tr>
<tr>
<td><code>defun om-getBind</code></td>
<td>1294</td>
</tr>
<tr>
<td><code>defun om-getBVar</code></td>
<td>1294</td>
</tr>
<tr>
<td><code>defun om-getByteArray</code></td>
<td>1295</td>
</tr>
<tr>
<td><code>defun om-getEndApp</code></td>
<td>1295</td>
</tr>
<tr>
<td><code>defun om-getEndAtp</code></td>
<td>1295</td>
</tr>
<tr>
<td><code>defun om-getEndAttr</code></td>
<td>1295</td>
</tr>
<tr>
<td><code>defun om-getEndBind</code></td>
<td>1296</td>
</tr>
<tr>
<td><code>defun om-getEndBVar</code></td>
<td>1296</td>
</tr>
<tr>
<td><code>defun om-getEndError</code></td>
<td>1296</td>
</tr>
<tr>
<td><code>defun om-getEndObject</code></td>
<td>1296</td>
</tr>
<tr>
<td><code>defun om-getError</code></td>
<td>1297</td>
</tr>
<tr>
<td><code>defun om-getFloat</code></td>
<td>1297</td>
</tr>
<tr>
<td><code>defun om-getError</code></td>
<td>1297</td>
</tr>
<tr>
<td><code>defun om-getInt</code></td>
<td>1297</td>
</tr>
<tr>
<td><code>defun om-getObject</code></td>
<td>1297</td>
</tr>
</tbody>
</table>
defun om-getString ................................. 1298
defun om-getSymbol .................................. 1298
defun om-getType .................................... 1298
defun om-getVar ..................................... 1298
defun om-putApp ..................................... 1299
defun om-putAtp ..................................... 1299
defun om-putAttr ................................... 1299
defun om-getVar ..................................... 1300
defun om-putByteArray ................................. 1300
defun om-putEndApp .................................. 1300
defun om-putEndAtp .................................. 1300
defun om-putEndAttr .................................. 1301
defun om-putEndBind .................................. 1301
defun om-putEndBVar .................................. 1301
defun om-putEndError .................................. 1301
defun om-putEndObject ................................ 1302
defun om-putError .................................... 1302
defun om-putFloat .................................... 1302
defun om-putInt ...................................... 1302
defun om-putObject ................................... 1303
defun om-putString ................................... 1303
defun om-putSymbol ................................... 1303
defun om-putVar ...................................... 1303
defun om-stringToStringPtr ............................. 1304
defun om-stringPtrToString .............................. 1304

defun makeByteWordVec2 .............................. 1305
defmacro spadConstant .................................. 1305

40 NRLIB code.lisp support code ........................ 1305

41 Monitoring execution ................................ 1307
defvar *monitor-domains* .................................. 1313
defvar *monitor-nrlibs* .................................. 1313
defvar *monitor-table* .................................. 1314
defstruct $monitor-data ................................ 1314
defstruct $libstream ................................... 1314
defun Initialize the monitor statistics hashtable .......... 1314
defun End the monitoring process, we cannot restart .......... 1315
defun Return a list of the monitor-data structures ............ 1315
defun Add a function to be monitored ........................ 1316
defun Remove a function being monitored ...................... 1316
defun Enable all (or optionally one) function for monitoring ........ 1316
defun Disable all (optionally one) function for monitoring .......... 1317
defun Reset the table count for the table (or a function) .......... 1317
defun Incr the count of fn by 1 ............................ 1318
defun Decr the count of fn by 1 ........................................ 1318
defun Return the monitor information for a function .............. 1319
defun Hang a monitor call on all of the defuns in a file ........... 1319
defun Return a list of the functions with zero count fields ...... 1319
defun Return a list of functions with non-zero counts ............ 1320
defun Write out a list of symbols or structures to a file .......... 1320
defun Save the *monitor-table* in loadable form .................. 1321
defun restore a checkpointed file .................................. 1321
defun Printing help documentation ................................ 1322
Monitoring algebra files .............................................. 1324
defun Monitoring algebra code.lsp files ............................. 1324
defun Monitor autoloaded files ....................................... 1324
defun Monitor an nrlib ............................................... 1325
defun Given a monitor-data item, extract the nrlib name .......... 1325
defun Is this an exposed algebra function? ......................... 1326
defun Monitor exposed domains ....................................... 1326
defun Generate a report of the monitored domains ................. 1327
defun Parse an )abbrev expression for the domain name ............ 1328
defun Given a spad file, report all nrlibs it creates .............. 1328
defun Print percent of functions tested ............................. 1329
defun Find all monitored symbols containing the string ............ 1329

42 HyperDoc Basic Command support ................................. 1331
  42.1 Hyperdoc macro handling and util.lit ........................ 1331
defvar $htMacroTable ............................................. 1331
defvar $primitiveHtCommands ..................................... 1331
defun Build the table of hyperdoc macros ......................... 1333
defun Get new command name and number of args .................... 1333
defun Is the first string a prefix of the second? ................. 1334
42.2 Functions creating pages .................................... 1335
defun Basic Command matrix entry ................................ 1336
defun Read Matrix .................................................. 1336
defun Input Matrix By Formula ................................... 1337
defun Basic Command Matrix by Formula generate .................. 1338
defun Input Explicit Matrix ....................................... 1339
defun Basic Command generate explicit matrix ...................... 1341
defun Basic Command generate matrix ................................ 1341
defun Basic Command iteration ..................................... 1342
defun Indefinite Integration Basic Command ......................... 1343
defun bcIndefiniteIntegrateGen ................................... 1344
defun Definite Integration Basic Command ......................... 1344
defun bcDefiniteIntegrateGen ..................................... 1345
defun Sum Basic Command ........................................... 1346
defun bcSumGen ..................................................... 1347
defun Sum Basic Command ........................................... 1348
defun bcProductGen .................................................. 1348
defun Differentiate Basic Command ........................................ 1349
defun bcDifferentiateGen .................................................. 1350
defun Draw Basic Command ................................................ 1351
defun Draw Basic Command by Function .................................. 1352
defun bcDraw2DfuncGen ..................................................... 1353
defun Draw Basic Command by Parameters ................................ 1353
defun bcDraw2DparGen ....................................................... 1355
defun Draw Basic Command by Equation Solution ....................... 1355
defun bcDraw2DSolveGen .................................................... 1356
defun Draw Basic Command by 3D function ................................ 1357
defun bcDraw3DfuncGen ..................................................... 1357
defun Draw Basic Command by 3D parameterized tube .................... 1360
defun bcDraw3DparGen ....................................................... 1361
defun Draw Basic Command by 3D parameterized function .............. 1362
defun bcDraw3Dpar1Gen ..................................................... 1363
defun Series Basic Command ............................................... 1364
defun Series Basic Command expand around a point ..................... 1365
defun bcSeriesExpansionGen ............................................... 1366
defun Series Basic Command series by formula ........................ 1366
defun Taylor Series Basic Command ...................................... 1367
defun bcSeriesByFormulaGen ............................................... 1369
defun Laurent Series Basic Command .................................... 1369
defun bcLaurentSeriesGen .................................................. 1371
defun Puiseux Series Basic Command .................................... 1371
defun bcTaylorSeriesGen .................................................... 1372
defun bcLaurentSeriesGen .................................................. 1373
defun bcPuiseuxSeriesGen .................................................. 1373
defun bcSeriesGen .......................................................... 1373
defun Limit Basic Command ................................................ 1374
defun Real Limit Basic Command .......................................... 1374
defun Real Limit Basic Command options ................................ 1376
defun bcRealLimitGen ....................................................... 1377
defun Complex Limit Basic Command ..................................... 1377
defun bcComplexLimitGen ................................................... 1379
defvar $systemType ......................................................... 1379
defvar $numberOfEquations ............................................... 1380
defvar $solutionMethod .................................................... 1380
defun Solve Basic Command ................................................. 1380
defun Linear Solve Basic Command ....................................... 1381
defun Linear Solve Equations Basic Command .......................... 1382
defun bcSystemSolve ....................................................... 1382
defun bcSolveSingle ......................................................... 1383
defun bcSystemSolveEqns1 .................................................. 1384
defun bcLinearSolveEqns1 .................................................. 1384
defun bcInputSolveInfo ..................................................... 1384
defun bcInputEquations .................................................... 1385
defun Create a variable string ............................................ 1387
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>defun bcMakeUnknowns</td>
<td>1387</td>
</tr>
<tr>
<td>defun bcMakeEquations</td>
<td>1388</td>
</tr>
<tr>
<td>defun bcMakeLinearEquations</td>
<td>1388</td>
</tr>
<tr>
<td>defun bcInputEquationsEnd</td>
<td>1389</td>
</tr>
<tr>
<td>defun bcSolveEquationsNumerically</td>
<td>1389</td>
</tr>
<tr>
<td>defun bcSolveNumerically1</td>
<td>1390</td>
</tr>
<tr>
<td>defun bcSolveEquations</td>
<td>1390</td>
</tr>
<tr>
<td>defun Linear Solve Basic Command trampoline</td>
<td>1391</td>
</tr>
<tr>
<td>defun Linear Solve Basic Command options</td>
<td>1391</td>
</tr>
<tr>
<td>defun bcLinearExtractMatrix</td>
<td>1392</td>
</tr>
<tr>
<td>defun Linear Solve Basic Command options</td>
<td>1392</td>
</tr>
<tr>
<td>defun bcLinearSolveMatrixInhomoGen</td>
<td>1393</td>
</tr>
<tr>
<td>defun bcLinearSolveMatrixHomo</td>
<td>1394</td>
</tr>
<tr>
<td>defun bcLinearMatrixGen</td>
<td>1394</td>
</tr>
<tr>
<td>defun linearFinalRequest</td>
<td>1395</td>
</tr>
<tr>
<td>defun explainLinear</td>
<td>1395</td>
</tr>
<tr>
<td>defun finalExactRequest</td>
<td>1395</td>
</tr>
<tr>
<td>defun bcLinearSolveEqnsGen</td>
<td>1396</td>
</tr>
<tr>
<td>defun bcGenEquations</td>
<td>1396</td>
</tr>
<tr>
<td>defun Output the final formula</td>
<td>1397</td>
</tr>
<tr>
<td>defun convert arguments into function call syntax</td>
<td>1397</td>
</tr>
<tr>
<td>defun bcString2HyString2</td>
<td>1398</td>
</tr>
<tr>
<td>defun bcString2HyString</td>
<td>1398</td>
</tr>
<tr>
<td>defun find a character position in a string</td>
<td>1398</td>
</tr>
<tr>
<td>defun Basic Command result page</td>
<td>1398</td>
</tr>
<tr>
<td>defun Basic Command result page – NAG version</td>
<td>1399</td>
</tr>
<tr>
<td>defun bcOptional</td>
<td>1399</td>
</tr>
<tr>
<td>defun create a vertical space on a page</td>
<td>1400</td>
</tr>
<tr>
<td>defun break a string into words</td>
<td>1400</td>
</tr>
<tr>
<td>defun format words into a string</td>
<td>1400</td>
</tr>
<tr>
<td>defun format a vector</td>
<td>1400</td>
</tr>
<tr>
<td>defun format an error message</td>
<td>1401</td>
</tr>
<tr>
<td>defun format intervals</td>
<td>1401</td>
</tr>
<tr>
<td>defun Basic Command page not ready</td>
<td>1401</td>
</tr>
<tr>
<td>defun pad a string with blanks</td>
<td>1402</td>
</tr>
<tr>
<td>defun construct a name string</td>
<td>1402</td>
</tr>
<tr>
<td>defun construct a name string</td>
<td>1402</td>
</tr>
<tr>
<td>defvar $bcParseOnly</td>
<td>1403</td>
</tr>
<tr>
<td>defvar $htLineList</td>
<td>1403</td>
</tr>
<tr>
<td>defvar $htLineList</td>
<td>1403</td>
</tr>
<tr>
<td>defvar $curpage</td>
<td>1403</td>
</tr>
<tr>
<td>defvar $activePageList</td>
<td>1403</td>
</tr>
<tr>
<td>defun htpDestroyPage</td>
<td>1403</td>
</tr>
<tr>
<td>HTPAGE STRUCTURE</td>
<td>1404</td>
</tr>
<tr>
<td>defun htpName</td>
<td>1404</td>
</tr>
<tr>
<td>defun htpSetName</td>
<td>1404</td>
</tr>
<tr>
<td>defun htpDomainConditions</td>
<td>1405</td>
</tr>
</tbody>
</table>
defun htpSetDomainConditions .......................... 1405
defun htpDomainVariableAlist .......................... 1405
defun htpSetDomainVariableAlist ....................... 1405
defun htpDomainPvarSubstList ........................ 1406
defun htpSetDomainPvarSubstList ....................... 1406
defun htpRadioButtonAlist ............................ 1406
defun htpButtonValue .................................. 1406
defun htpSetRadioButtonAlist .......................... 1407
defun htpInputAreaAlist ............................... 1407
defun htpSetInputAreaAlist ............................ 1407
defun htpAddInputAreaProp ............................. 1408
defun htpPropertyList ................................. 1408
defun htpProperty ..................................... 1408
defun htpSetProperty ................................... 1408
defun htpLabelInputString .............................. 1409
defun htpLabelFilteredInputString ..................... 1409
defun replacePercentByDollar, fn ..................... 1409
defun replacePercentByDollar ........................... 1410
defun htpSetLabelInputString ........................... 1410
defun htpLabelSpadValue ................................ 1410
defun htpSetLabelSpadValue ............................. 1410
defun htpLabelErrorMsg ................................. 1411
defun htpSetLabelErrorMsg .............................. 1411
defun htpLabelType ..................................... 1411
defun htpLabelDefault .................................. 1411
defun htpLabelSpadType ................................ 1412
defun htpLabelFilter ................................... 1412
defun htpPageDescription ............................... 1412
defun htpSetPageDescription ............................ 1413
defun htpAddToPageDescription ......................... 1413
defun issue a single hypertex line or group of lines .. 1413
defun bcHt ........................................... 1414
defun bcIssueHt ........................................ 1414
defun mapStringize ...................................... 1414
defun basicStringize .................................... 1415
defun stringize ......................................... 1415
defun htInitPage ....................................... 1415
defun htAddHeading ...................................... 1415
defun htShowPage ....................................... 1416
defun show the page which has been computed .......... 1416
defun make a page given the description in itemList .. 1416
defun htMakePage1 ....................................... 1417
defun htMakeErrorPage ................................... 1418
defun htQuote ........................................... 1418
defun htProcessToggleButtons ......................... 1419
defun htProcessBcButtons ............................... 1420
defun htProcessBeStrings ........................................ 1421
defun bcSadFaces .................................................. 1422
defun htLispLinks .................................................. 1422
defun htLispMemoLinks ............................................. 1423
defun htBeLinks .................................................... 1423
defun htBeLispLinks ............................................... 1424
defun beforeAfter ................................................ 1425
defun mkCurryFun .................................................. 1425
defun htRadioButtons ............................................. 1426
defun htBeRadioButtons ........................................... 1427
defun setUpDefault ............................................... 1428
defun buttonNames ................................................. 1429
defun htInputStrings .............................................. 1429
defun htProcessDomainConditions ............................... 1431
defun renamePatternVariables .................................... 1431
defun renamePatternVariables1 ................................. 1431
defun substFromAlist ............................................. 1433
defun computeDomainVariableAlist ............................. 1433
defun pvarCondList ............................................... 1434
defun pvarCondList1 ............................................... 1434
defun pvarsOfPattern ............................................. 1435
defun htMakeTemplates,substLabel ............................. 1435
defun htMakeTemplates ............................................ 1436
defun templateParts ............................................... 1437
defun htMakeDoneButton ......................................... 1437
defun htProcessDoneButton ...................................... 1438
defun htMakeButton ................................................ 1438
defun bchtMakeButton ............................................. 1440
defun htProcessDoitButton ..................................... 1441
defun htMakeDoitButton ......................................... 1441
defun doDoitButton ................................................. 1442
defun executeInterpreterCommand ............................... 1442
defun htDoneButton ............................................... 1442
defun typeCheckInputAreas ...................................... 1443
defun checkCondition ............................................. 1445
defun condErrorMsg ............................................... 1446
defun parseAndEval ............................................... 1447
defun parseAndEval1 .............................................. 1447
defun oldParseString ............................................. 1448
defun makeSpadCommand .......................................... 1448
defun htMakeInputList ........................................... 1448
defun bracketString ............................................... 1449
defun quoteString ................................................ 1449
defivar $funnyQuote .............................................. 1449
defivar $funnyBacks .............................................. 1450
defun htEscapeString ............................................. 1450
| defun htsv                          | 1450 |
| defun httSetVars                  | 1450 |
| defun httShowSetTree              | 1451 |
| defun httShowCount                | 1453 |
| defun httShowSetTreeValue         | 1453 |
| defun mkSetTitle                  | 1454 |
| defun listofStrings2String        | 1454 |
| defun httShowSetPage              | 1454 |
| defun httShowLiteralsPage         | 1455 |
| defun httSetLiterals              | 1455 |
| defun httSetLiteral               | 1456 |
| defun httShowIntegerPage          | 1457 |
| defun httSetInteger               | 1458 |
| defun httShowFunctionPage         | 1458 |
| defun httShowFunctionPageContinued| 1459 |
| defun httSetvarDoneButton         | 1460 |
| defun httFunctionSetLiteral       | 1460 |
| defun httSetFunCommand            | 1461 |
| defun httSetFunCommandContinue    | 1461 |
| defun httKill                      | 1461 |
| defun httSetNotAvailable          | 1462 |
| defun httDoNothing                | 1463 |
| defun httCheck                     | 1463 |
| defun parseWord                   | 1463 |
| defun httCheckList                | 1464 |
| defun translateYesNoToTrueFalse   | 1465 |
| defun chkNameList                 | 1465 |
| defun chkPosInteger               | 1466 |
| defun chkOutputFileName           | 1466 |
| defun chkDirectory                | 1466 |
| defun chkNonNegativeInteger       | 1467 |
| defun chkRange                    | 1467 |
| defun chkAllNonNegativeInteger    | 1467 |
| defun httMakePathKey, fn          | 1468 |
| defun httMakePathKey              | 1468 |
| defun httMarkTree                 | 1468 |
| defun httSetHistory               | 1469 |
| defun httSetOutputLibrary         | 1469 |
| defun httSetInputLibrary          | 1469 |
| defun httSetExpose                | 1470 |
| defun httSetOutputCharacters      | 1470 |
| defun httSetLinkerArgs            | 1470 |
| defun httSetCache                 | 1470 |
| defun httCacheAddChoice           | 1471 |
| defun httMakeLabel                | 1472 |
| defun httCacheSet                 | 1473 |
defun htAllOrNum ........................................... 1474
defun htCacheOne ......................................... 1474
defvar $historyDisplayWidth ............................... 1475
defvar $newline ............................................. 1475
defun downlink ............................................. 1475
defun dbNonEmptyPattern .................................. 1476
defun htSystemVariables,gn ................................ 1476
defun htSystemVariables,fn ............................... 1477
defun htSystemVariables,displayOptions ................. 1477
defun htSystemVariables,functionTail .................... 1478
defun htSystemVariables .................................... 1479
defun htSetSystemVariableKind ......................... 1481
defun htSetSystemVariable ................................ 1482
defun htGloss .............................................. 1482
defun htGlossPage ........................................... 1482
defun gatherGlossLines .................................... 1485
defun htGlossSearch ....................................... 1486
defun htGreekSearch ....................................... 1487
defun htTextSearch ........................................ 1489
defun htTutorialSearch .................................... 1491
defun mkUnixPattern .................................... 1492

43 Browser Support Code 1493
43.1 Pages Initiated from HyperDoc Pages .................. 1493
Search routines ............................................. 1493
defun dKind ................................................... 1493
defun checkFilter .......................................... 1493
defun Concatenate words with blanks .................... 1494
defun Make constructor names lowercase .................. 1494
defun string2Constructor .................................. 1494
defvar dbDelimiters ........................................ 1495
defun String to words respecting delimiters ............. 1495
defun Next word respecting delimiters .................. 1496
defun Hyperdoc category search ......................... 1496
defun Hyperdoc default domain search .................... 1499
defun Hyperdoc domain search .............................. 1499
defun Hyperdoc package search ................................ 1499
defun Hyperdoc constructor search ....................... 1500
defun Hyperdoc default constructor search ............... 1500
defun Read libdb.text at file-position n ................ 1500
defun String trim with newlines removed .................. 1500
defun Hyperdoc common constructor search ............... 1501
defun conSpecialString? .................................... 1502
Page construction ......................................... 1503
defun conPage ............................................... 1503
defun gets line quickly for constructor name or abbreviation 1504
defun conPageConEntry ........................................ 1504
defun kdPageInfo ............................................. 1505
defun kArgPage ................................................ 1506
defun mkDomTypeForm ......................................... 1507
defun domainDescendantsOf ................................. 1507
43.2 Branches of Constructor Page ......................... 1508
defun kiPage .................................................. 1508
defun kePage .................................................. 1509
defun kePageOpAlist ......................................... 1511
defun kePageDisplay ......................................... 1512
defun ksPage .................................................. 1513
defun dbSearchOrder ......................................... 1514
defun kcPage .................................................. 1515
defun kcpPage ................................................ 1518
defun reduceAlistForDomain ............................... 1519
defun kcaPage ................................................ 1520
defun kcdPage ................................................ 1520
defun kcdoPage ............................................... 1521
defun kcaPage1 ............................................... 1521
defun kccPage ................................................ 1522
defun augmentHasArgs ....................................... 1523
defun kcdePage ............................................... 1524
defun getDependentsOfConstructor ........................ 1524
defun kcuPage ................................................ 1525
defun getUsersOfConstructor ............................... 1526
defun kcnPage ................................................ 1526
defun koPageInputAreaUnchanged? ......................... 1527
defun kDomainName .......................................... 1527
defun kArgumentCheck ...................................... 1528
defun dbMkEvalable .......................................... 1529
defun topLevelInterpEval .................................. 1530
defun kisValidType ......................................... 1530
defun kCheckArgumentNumbers ............................. 1531
defun parseNoMacroFromString ............................. 1531
defun mkConform ............................................. 1532
43.3 Operation Page for a Domain Form from Scratch .... 1532
defun conOpPage ............................................. 1532
defun conOpPage1 ........................................... 1533
defun dbCompositeWithMap .................................. 1534
defun dbExtractUnderlyingDomain ......................... 1535
43.4 Operation Page from Main Page ...................... 1535
defun koPage ................................................. 1535
defun koPageFromKKPage .................................... 1536
defun koPageAux ............................................. 1536
defun koPageAux1 ............................................ 1537
defun koaPageFilterByName ................................. 1537
46 The Global Variables

46.1 Star Global Variables

*eof* ......................................................... 1609
*features* .................................................. 1609
*package* .................................................... 1609
*standard-input* ........................................... 1610
*standard-output* ......................................... 1610
*top-level-hook* ............................................ 1610

46.2 Dollar Global Variables

$boot ......................................................... 1613
coerceFailure .............................................. 1613
$currentLine ............................................... 1613
$displayStartMsgs ......................................... 1613
$erMsgToss .................................................. 1613
$frameRecord ............................................... 1613
$intRestart .................................................. 1613
$intTopLevel ............................................... 1614
$IOindex ..................................................... 1614
$lastPos ...................................................... 1614
$libQuiet .................................................... 1614
$msgDatabaseName .......................................... 1614
$ncMsgList .................................................. 1614
$newcompErrorCount ....................................... 1614
$nopos ....................................................... 1615
$oldHistoryFileName ....................................... 1615
$okToExecuteMachineCode .................................. 1615
$options ..................................................... 1615
$previousBindings .......................................... 1615
$reportundo .................................................. 1615
$spad ........................................................ 1615
$SpadServer .................................................. 1616
$SpadServerName ............................................ 1616
$systemCommandFunction .................................... 1616
$top_level .................................................... 1616
$quitTag ....................................................... 1616
$useInternalHistoryTable ................................... 1616

47 Signatures .................................................. 1617

48 Bibliography ............................................... 1619

49 Index ....................................................... 1623
New Foreword

On October 1, 2001 Axiom was withdrawn from the market and ended life as a commercial product. On September 3, 2002 Axiom was released under the Modified BSD license, including this document. On August 27, 2003 Axiom was released as free and open source software available for download from the Free Software Foundation’s website, Savannah.

Work on Axiom has had the generous support of the Center for Algorithms and Interactive Scientific Computation (CAISS) at City College of New York. Special thanks go to Dr. Gilbert Baumslag for his support of the long term goal.

The online version of this documentation is roughly 1000 pages. In order to make printed versions we’ve broken it up into three volumes. The first volume is tutorial in nature. The second volume is for programmers. The third volume is reference material. We’ve also added a fourth volume for developers. All of these changes represent an experiment in print-on-demand delivery of documentation. Time will tell whether the experiment succeeded.

Axiom has been in existence for over thirty years. It is estimated to contain about three hundred man-years of research and has, as of September 3, 2003, 143 people listed in the credits. All of these people have contributed directly or indirectly to making Axiom available. Axiom is being passed to the next generation. I’m looking forward to future milestones.

With that in mind I’ve introduced the theme of the “30 year horizon”. We must invent the tools that support the Computational Mathematician working 30 years from now. How will research be done when every bit of mathematical knowledge is online and instantly available? What happens when we scale Axiom by a factor of 100, giving us 1.1 million domains? How will we integrate theory with code? How will we integrate theorems and proofs of the mathematics with space-time complexity proofs and running code? What visualization tools are needed? How do we support the conceptual structures and semantics of mathematics in effective ways? How do we support results from the sciences? How do we teach the next generation to be effective Computational Mathematicians?

The “30 year horizon” is much nearer than it appears.

Tim Daly
CAISS, City College of New York
November 10, 2003 ((iHy))
Chapter 1

The Interpreter

The Axiom interpreter is a large common lisp program. It has several forms of interaction and run from terminal in a standalone fashion, run under the control of a session handler program, run as a web server, or run in a unix pipe.
Chapter 2

The Fundamental Data Structures

2.1 Frames and the Interpreter Frame Ring

Axiom has the notion of "frames". A frame is a data structure which holds all the vital data from an Axiom session.

The list of frames is structured as a ring. New frames can be added which will hold computations of independent information. The interpreter \texttt{frame} command allows operations on frames. From the command line the user can create, modify, change, and delete frames.

2.2 \texttt{frame} Command

\texttt{frame man page}

— frame.help —

====================================================================
A.11. frame
====================================================================

User Level Required: interpreter

Command Syntax:

- \texttt{)frame new frameName}
- \texttt{)frame drop [frameName]}
- \texttt{)frame next}
- )frame last
- )frame names
- )frame import frameName [objectName1 [objectName2 ...]]
- )set message frame on | off
- )set message prompt frame

Command Description:

A frame can be thought of as a logical session within the physical session that you get when you start the system. You can have as many frames as you want, within the limits of your computer’s storage, paging space, and so on. Each frame has its own step number, environment and history. You can have a variable named a in one frame and it will have nothing to do with anything that might be called a in any other frame.

Some frames are created by the HyperDoc program and these can have pretty strange names, since they are generated automatically. To find out the names of all frames, issue

)frame names

It will indicate the name of the current frame.

You create a new frame ‘‘quark’’ by issuing

)frame new quark

The history facility can be turned on by issuing either )set history on or )history )on. If the history facility is on and you are saving history information in a file rather than in the AXIOM environment then a history file with filename quark.axh will be created as you enter commands. If you wish to go back to what you were doing in the ‘‘initial’’ frame, use

)frame next

or

)frame last

to cycle through the ring of available frames to get back to ‘‘initial’’.

If you want to throw away a frame (say ‘‘quark’’), issue

)frame drop quark

If you omit the name, the current frame is dropped.

If you do use frames with the history facility on and writing to a file, you may want to delete some of the older history files. These are directories, so you may want to issue a command like rm -r quark.axh to the operating system.
2.2. \texttt{)FRAME COMMAND}

You can bring things from another frame by using \texttt{)frame import}. For example, to bring the \texttt{f} and \texttt{g} from the frame \texttt{"quark"} to the current frame, issue

\texttt{)frame import quark f g}

If you want everything from the frame \texttt{"quark"}, issue

\texttt{)frame import quark}

You will be asked to verify that you really want everything.

There are two \texttt{)set} flags to make it easier to tell where you are.

\texttt{)set message frame on | off}

will print more messages about frames when it is set on. By default, it is off.

\texttt{)set message prompt frame}

will give a prompt that looks like

\texttt{initial (1) ->}

when you start up. In this case, the frame name and step make up the prompt.

Also See:
\texttt{o )history}
\texttt{o )set}

---

1

The frame mechanism uses several dollar variables.

Primary variables are those which exist solely to make the frame mechanism work.

The $interpreterFrameName contains a symbol which is the name of the current frame in use.

The $interpreterFrameRing contains a list of all of the existing frames. The first frame on the list is the “current” frame. When Axiom is started directly there is only one frame named “initial”.

If the system is started under sman (using the axiom shell script, for example), there are two frames, “initial” and “frame0”. In this case, “frame0” is the current frame. This can cause subtle problems because functions defined in the axiom initialization file (.axiom.input) will be defined in frame “initial” but the current frame will be “frame0”. They will appear to be

\footnote{“history” (25.23 p 821) “set” (25.51 p 1013)}
undefined. However, if the user does "frame next" they can switch to the "initial" frame and see the functions correctly defined.

The $frameMessages variable controls when frame messages will be displayed. The variable is initially NIL. It can be set on (T) or off (NIL) using the system command:

)set message frame on | off

Setting frame messages on will output a line detailing the current frame after every output is complete.

See the set message frame(p945) setmessageframe section for more details.

The frame collects and uses a few top level variables. These are: $InteractiveFrame, $IOindex, $HiFiAccess, $HistList, $HistListLen, $HistListAct, $HistRecord, $internalHistoryTable, and $localExposureData.

These variables can also be changed by the frame mechanism when the user requests changing to a different frame.

2.3 Data Structures

The interpreter information is kept in a frame which is a 10 part data structure of the form (see 2.5). The parts of a frame and their initial, default values are:

1. $interpreterFrameName(p23) interpreterFrameName, a string, is the name of the current frame.
2. $InteractiveFrame(p23) InteractiveFrame which defaults to (nil)
3. $IOindex(p24) IOindex also known as the step number, which defaults to 1
4. $HiFiAccess(p937) HiFiAccess
5. $HistList(p31) HistList
6. $HistListLen(p32) HistListLen
7. $HistListAct(p32) HistListAct
8. $HistRecord(p32) HistRecord
9. $internalHistoryTable(p32) internalHistoryTable which defaults to nil
10. $localExposureDataDefault(p99) is a copy of the current local exposure data

There are a set of functions to manipulate frames. The internal set of frame functions are

- initializeInterpreterFrameRing(p12) creates the original frame ring, inserts an initial frame, and updates all the global variables from the initial frame.
2.4. FRAME ACCESS MACROS

- **emptyInterpreterFrame**(p12) creates a new, empty frame.
- **createCurrentInterpreterFrame**(p14) collects the environment into a frame.
- **updateFromCurrentInterpreterFrame**(p15) sets all of the global variables from the current frame.
- **frameEnvironment**(p16) returns the frameInteractive component of a named frame or a new, empty environment.
- **findFrameInRing**(p16) given the name, find the named frame
- **updateCurrentInterpreterFrame**(p16) collects the normal contents of the world into a frame object, places it first on the frame list, and then sets the current values of the world from the frame object.
- **nextInterpreterFrame**(p17) updates the current frame to make sure all of the current information is recorded. If there are more frame elements in the list then this will destructively move the current frame to the end of the list, that is, assume the frame list reads (1 2 3) this function will destructively change it to (2 3 1).
- **previousInterpreterFrame**(p18) moves to the previous frame in the ring.
- **changeToNamedInterpreterFrame**(p17) change to the named frame.
- **addNewInterpreterFrame**(p18) update the current frame, initialize the history, make a new empty frame, and initialize all of the global variables from the empty frame.
- **closeInterpreterFrame**(p22) when there is more than one frame, delete the current frame and initialize all the global variables from the next frame in the ring.
- **displayFrameNames**(p13) print all the frame names and indicate which one is the current frame.
- **importFromFrame**(p19) imports items from a different frame into the current frame

2.4 Frame Access Macros

First Frame Component – frameName

defmacro frameName

--- defmacro frameName 0 ---

(defmacro frameName (frame)
  `(first ,frame))
Second Frame Component – frameInteractive

\textbf{defmacro frameInteractive}

--- \textbf{defmacro frameInteractive 0} ---

\begin{verbatim}
(defmacro frameInteractive (frame)
  '(second ,frame))
\end{verbatim}

---

Third Frame Component – frameIOIndex

\textbf{defmacro frameIOIndex}

--- \textbf{defmacro frameIOIndex 0} ---

\begin{verbatim}
(defmacro frameIOIndex (frame)
  '(third ,frame))
\end{verbatim}

---

Fourth Frame Component – frameHiFiAccess

\textbf{defmacro frameHiFiAccess}

--- \textbf{defmacro frameHiFiAccess 0} ---

\begin{verbatim}
(defmacro frameHiFiAccess (frame)
  '(fourth ,frame))
\end{verbatim}

---

Fifth Frame Component – frameHistList

\textbf{defmacro frameHistList}

--- \textbf{defmacro frameHistList 0} ---
2.4. FRAME ACCESS MACROS

(defmacro frameHistList (frame)
  '(fifth ,frame))

Sixth Frame Component – frameHistListLen

defmacro frameHistListLen

  — defmacro frameHistListLen 0 —

(defmacro frameHistListLen (frame)
  '(sixth ,frame))

Seventh Frame Component – frameHistListAct

defmacro frameHistListAct

  — defmacro frameHistListAct 0 —

(defmacro frameHistListAct (frame)
  '(seventh ,frame))

Eighth Frame Component – frameHistRecord

defmacro frameHistRecord

  — defmacro frameHistRecord 0 —

(defmacro frameHistRecord (frame)
  '(eighth ,frame))

Ninth Frame Component – frameHistoryTable
2.5 Functions to manipulate frames

The top level frame command

(defun |frame| (l)
  "The top level frame command"
  (|frameSpad2Cmd| l))
2.5. FUNCTIONS TO MANIPULATE FRAMES

(defun frameSpad2Cmd (args)
  "The top level frame command handler"
  (let (frameArgs arg a)
    (declare (special $options$))
    (setq frameArgs '(drop import last names new next))
    (cond
      ($options$)
      (throwKeyedMsg "The %1 system command takes arguments but no options." (cons "frame" nil)))
      (null args) (helpSpad2Cmd (cons 'frame nil)))
      (t
       (setq arg (selectOptionLC (car args) frameArgs 'optionError))
       (setq args (cdr args))
       (when (and (consp args)
                   (eq (qcdr args) nil)
                   (progn (setq a (qcar args)) t))
         (setq args a))
       (when (atom args) (setq args (object2Identifier args)))
       (case arg
         (drop
          (if (and args (consp args))
            (throwKeyedMsg "%1 is not a valid frame name." (cons args nil))
            (closeInterpreterFrame args)))
         (import (importFrame args))
         (last (previousInterpreterFrame))
         (names (displayFrameNames))
         (new
          (if (and args (consp args))
            (throwKeyedMsg "%1 is not a valid frame name.")))
CHAPTER 2. THE FUNDAMENTAL DATA STRUCTURES

(cons args nil)
(|addNewInterpreterFrame| args))
(|next| (|nextInterpreterFrame|))
(t nil)))))))

Initializing the Interpreter Frame Ring

There can be multiple frames and these live in a top-level variable called \texttt{$\textbf{interpreterFrameRing}$}. This variable holds a circular list of frames.

This function creates an empty, initial frame named “initial” and creates a list of frames containing an empty frame. This list is the interpreter frame ring and is not actually circular but is managed as a circular list.

As a final step we update the world from this frame. This has the side-effect of resetting all the important global variables to their initial values.

[emptyInterpreterFrame p12]
[updateFromCurrentInterpreterFrame p15]
[$\textbf{interpreterFrameName}$ p23]
[$\textbf{interpreterFrameRing}$ p23]

— defun initializeInterpreterFrameRing —

(defun |initializeInterpreterFrameRing| ()
  "Initializing the Interpreter Frame Ring"
  (declare (special |$\textbf{interpreterFrameName}$| |$\textbf{interpreterFrameRing}$|))
  (setq |$\textbf{interpreterFrameName}$| '|initial|)
  (setq |$\textbf{interpreterFrameRing}$|
    (1ist (|emptyInterpreterFrame| |$\textbf{interpreterFrameName}$|)))
  (|updateFromCurrentInterpreterFrame|)
  nil)

Create a new, empty Interpreter Frame

[$\textbf{HiFiAccess}$ p937]
[$\textbf{HistList}$ p31]
[$\textbf{HistListLen}$ p32]
[$\textbf{HistListAct}$ p32]
[$\textbf{HistRecord}$ p32]
[$\textbf{localExposureDataDefault}$ p99]
2.5. FUNCTIONS TO MANIPULATE FRAMES

--- defun emptyInterpreterFrame 0 ---

(defun emptyInterpreterFrame (name)
  "Create a new, empty Interpreter Frame"
  (declare (special $HiFiAccess $HistList $HistListLen $HistListAct
    $HistRecord $localExposureDataDefault))
  (list name ; frame name
    (list (list nil)) ; environment
    1 ; $IOindex
    $HiFiAccess
    $HistList
    $HistListLen
    $HistListAct
    $HistRecord
    nil ; $internalHistoryTable
    (copy-seq $localExposureDataDefault)); $localExposureData

Create a list of all of the frame names

This function simply walks across the frame in the frame ring and returns a list of the name of each frame. [interpreterFrameRing p23]

--- defun frameNames 0 ---

(defun frameNames ()
  "Creating a List of all of the Frame Names"
  (declare (special $interpreterFrameRing))
  (mapcar #'(lambda (f) (frameName f)) $interpreterFrameRing))

Display the frame name list message

[bright p??]
[framename p??]
[interpreterFrameRing p23]

--- defun displayFrameNames 0 ---

(defun displayFrameNames ()
  "Display the Frame Names"
  (declare (special $interpreterFrameRing)))
(format t " The names of the existing frames are:~
 (format t " ~a ~
 (format t " The current frame is the first one listed.~")

---

Collect the global variables into a Frame

We can collect up all the current environment information into one frame element with this call. It creates a list of the current values of the global variables and returns this as a frame element.

(defun |createCurrentInterpreterFrame| 0 |
defun createCurrentInterpreterFrame| () |
"Collecting up the Environment into a Frame"
 (declare (special |$interpreterFrameName| |$InteractiveFrame| |$IOindex| |$HiFiAccess| |$HistList| |$HistListLen| |$HistListAct| |$HistRecord| |$internalHistoryTable| |$localExposureData|)
 (list  |
 (list  |
 (list  |
 (list  |
 (list  |
 (list  |
 (list  |

---
Update global variables from the Current Frame

The frames are kept on a circular list. The first element on that list is known as “the current frame”. This will initialize all of the interesting interpreter data structures from that frame.

```lisp
(defun updateFromCurrentInterpreterFrame ()
  "Update from the Current Frame"
  (let ((tmp1)
        (declare (special $interpreterFrameRing $interpreterFrameName $InteractiveFrame $IOindex $HiFiAccess $HistList $HistListLen $HistListAct $HistRecord $internalHistoryTable $localExposureData $frameMessages))
    (setq tmp1 (first $interpreterFrameRing))
    (setq $interpreterFrameName (frameName tmp1))
    (setq $InteractiveFrame (frameInteractive tmp1))
    (setq $IOindex (frameIOIndex tmp1))
    (setq $HiFiAccess (frameHiFiAccess tmp1))
    (setq $HistList (frameHistList tmp1))
    (setq $HistListLen (frameHistListLen tmp1))
    (setq $HistListAct (frameHistListAct tmp1))
    (setq $HistRecord (frameHistRecord tmp1))
    (setq $internalHistoryTable (frameHistoryTable tmp1))
    (setq $localExposureData (frameExposureData tmp1))
    (when $frameMessages
      (format t " Current interpreter frame is called ~a" $interpreterFrameName)))
```

---
Replace the current frame and update from the globals

This function collects the normal contents of the world into a frame object, places it first on
the frame list, and then sets the current values of the world from the frame object.

(defun updateCurrentInterpreterFrame ()
  "Update the Current Interpreter Frame"
  (declare (special $interpreterFrameRing))
  (rplaca $interpreterFrameRing (createCurrentInterpreterFrame))
  (updateFromCurrentInterpreterFrame))

Get Named Frame Environment (aka Interactive)

If the frame is found we return the environment portion of the frame otherwise we construct
an empty environment and return it. The initial values of an empty frame are created here.
This function returns a single frame that will be placed in the frame ring.

(defun frameEnvironment (fname)
  "Get Named Frame Environment (aka Interactive)"
  (let ((frame (findFrameInRing fname)))
    (if frame
      (frameInteractive frame)
      (list (list nil)))))

Find a Frame in the Frame Ring by Name

Each frame contains its name as the 0th element. We simply walk all the frames and if we
find one we return it. [boot-equal p7]

(defun findFrameInRing 0 ---
2.5. FUNCTIONS TO MANIPULATE FRAMES

(defun findFrameInRing (name)
  "Find a Frame in the Frame Ring by Name"
  (declare (special $interpreterFrameRing))
  (dolist (frame $interpreterFrameRing)
    (when (eq (frameName frame) name) (return frame))))

— Change to the Named Interpreter Frame —

(defun changeToNamedInterpreterFrame (name)
  "Change to the Named Interpreter Frame"
  (let (frame)
    (declare (special $interpreterFrameRing))
    (updateCurrentInterpreterFrame)
    (setq frame (findFrameInRing name))
    (when frame
      (setq $interpreterFrameRing (cons frame (delete $interpreterFrameRing frame :test #'equal)))
      (updateFromCurrentInterpreterFrame))))

— Move to the next Interpreter Frame in Ring —

(defun nextInterpreterFrame ()
  "Move to the next Interpreter Frame in Ring"
  (declare (special $interpreterFrameRing))
  (updateFromCurrentInterpreterFrame))
(when (cdr $interpreterFrameRing)
  (setq $interpreterFrameRing
    (nconc (cdr $interpreterFrameRing) (list (car $interpreterFrameRing)))))

Move to the previous Interpreter Frame in Ring

(updateCurrentInterpreterFrame p16)
(updateFromCurrentInterpreterFrame p15)
$interpreterFrameRing p23

— defun previousInterpreterFrame —

(defun previousInterpreterFrame ()
  "Move to the previous Interpreter Frame in Ring"
  (let (tmp1 l b)
    (declare (special $interpreterFrameRing))
    (updateCurrentInterpreterFrame)
    (when (cdr $interpreterFrameRing)
      (setq tmp1 (reverse $interpreterFrameRing))
      (setq l (car tmp1))
      (setq b (nreverse (cdr tmp1)))
      (setq $interpreterFrameRing (nconc (cons l nil) b))
      (updateFromCurrentInterpreterFrame)))

Add a New Interpreter Frame

(boot-equal p??)
(framename p??)
(throwKeyedMsg p??)
(updateCurrentInterpreterFrame p16)
(initHistList p821)
(emptyInterpreterFrame p12)
(updateFromCurrentInterpreterFrame p15)
(erase p??)
(histFileName p820)
$interpreterFrameRing p23

— defun addNewInterpreterFrame —
2.5. FUNCTIONS TO MANIPULATE FRAMES

(defvar $interpreterFrameRing)

(defun addNewInterpreterFrame (name)
  "Add a New Interpreter Frame"
  (declare (special $interpreterFrameRing))
  (if (null name)
      (throwKeyedMsg "You must provide a name for the new frame." nil)
      (progn
        (updateCurrentInterpreterFrame)
        (dolist (f $interpreterFrameRing)
          (when (eq name (frameName f)); existing frame with same name
            (throwKeyedMsg
              (format nil
                " You cannot use the name %1 for a new frame because an existing ~
                frame already has that name."
              )
            (list name))
            (initHistList)))
        (setq $interpreterFrameRing
          (cons (emptyInterpreterFrame name) $interpreterFrameRing))
        (updateFromCurrentInterpreterFrame)
        ($erase (histFileName))))

Import items from another frame

    [member p1198]
    [frameNames p13]
    [throwKeyedMsg p??]
    [boot-equal p??]
    [framename p??]
    [frameEnvironment p16]
    [upcase p1206]
    [queryUserKeyedMsg p??]
    [string2id-n p??]
    [importFromFrame p19]
    [sayKeyedMsg p27]
    [clearCmdParts p773]
    [seq p??]
    [exit p??]
    [putHist p832]
    [get p??]
    [getalist p??]
    [$interpreterFrameRing p23]

    — defun importFromFrame —

(defun importFromFrame (args)
"Import items from another frame"

(prog (templ1 fname x v props vars plist prop val m)
 (declare (special |$interpreterFrameRing|))
 (when (and args (atom args)) (setq args (cons args nil)))
 (if (null args)
   (throwKeyedMsg
    (format nil "))frame import must be followed by the frame name. The names~
of objects in that frame can then optionally follow the frame name." ~
    For example,
    %-ceon )frame import calculus %-ceoff ~
    imports all objects in the calculus frame, and ~
    %-ceon )frame import calculus epsilon delta %-ceoff ~
    imports the objects named epsilon and delta from the ~
    frame calculus. ~
    Please note that if the current frame contained any information ~
    about objects with these names, then that information would be ~
cleared before the import took place.")
 nil)
 (progn
 (setq templ1 args)
 (setq fname (car templ1))
 (setq args (cdr templ1))
 (cond
   ((null (member fname |frameNames|)) )
     (throwKeyedMsg
      (format nil " You cannot import anything from the frame %1 because ~
        that is not the name of an existing frame.")
      (cons fname nil)))
   ((boot-equal fname (frameName (car |$interpreterFrameRing|)))
     (throwKeyedMsg
      "You cannot import from the current frame (nor is there a need!)."))
   nil))
 (t
 (setq fenv (|frameEnvironment| fname))
 (cond
   ((null args)
    (setq x
     (upcase (queryUserKeyedMsg
      (format nil "Do you really want to import everything from the ~
        frame %1? If so, please enter y or yes :")
      (cons fname nil)))))
   (cond
    ((member (string2id-n x 1) '(y yes))
     (setq vars nil)
     (do ((tmp0 (caar fenv) (cdr tmp0)) (tmp1 nil))
     ((or (atom tmp0)
       (progn (setq tmp1 (car tmp0)) nil)
       (progn
        (progn
         (setq v (car tmp1))
         ...)
(setq props (cdr tmp1))
(tmp1)
(nil)
(_cond
((eq v '|--macros|)
 (do ((tmp2 props (cdr tmp2))
     (tmp3 nil))
     ((or (atom tmp2)
         (progn (setq tmp3 (car tmp2)) nil)
         (progn
         (progn
         (setq m (car tmp3)) tmp3)
        nil))
     (setq vars (cons m vars)))
     (t (setq vars (cons v vars))))))
(importFromFrame (cons fname vars)))
(t (sayKeyedMsg "AXIOM will not import everything from frame %1.
(cons fname nil))))
(t
do ((tmp4 args (cdr tmp4)) (v nil))
  ((or (atom tmp4) (progn (setq v (car tmp4)) nil)) nil)
  (seq
    (exit
      (progn
        (setq plist (getalist (caar fenv) v))
        (cond
          (plist
            (|clearCmdParts| (cons '|propert| (cons v nil)))
            (do ((tmp5 plist (cdr tmp5)) (tmp6 nil))
                ((or (atom tmp5)
                    (progn (setq tmp6 (car tmp5)) nil)) nil)
                (progn
                (progn
                (setq prop (car tmp6))
                (setq val (cdr tmp6))
                (setq val (cdr tmp6))
                (tmp6)
                nil))
            nil))
          (seq
            (exit (|putHist| v prop val |$InteractiveFrame|))))))
          (setq m (|get| '|--macros--| v fenv))
          (|putHist| '|--macros--| v m |$InteractiveFrame|))
         (t
        (sayKeyedMsg
          (format nil "AXIOM cannot import %1 from frame %2 because ~
            it cannot be found.")
        (cons v (cons fname nil)))))
        (sayKeyedMsg)
(format nil "Import from frame %1 is complete. Please issue ~
  \)display all if you wish to see the contents of ~
  \the current frame.")
(cons fname nil))))))))}})

---

Close an Interpreter Frame

[framename p??]
[throwKeyedMsg p??]
[$erase p??]
[makeHistFileName p819]
[updateFromCurrentInterpreterFrame p15]
[$interpreterFrameRing p23]
[$interpreterFrameName p23]

— defun closeInterpreterFrame —

(defun closeInterpreterFrame (name)
  "Close an Interpreter Frame"
  (declare (special $interpreterFrameRing $interpreterFrameName))
  (let (ifr found)
    (if (null (cdr $interpreterFrameRing))
      (if (and name (not (equal name $interpreterFrameName)))
        (throwKeyedMsg)
          (format nil "There is only one frame active and therefore that ~
            cannot be closed. Furthermore, the frame name you gave is not ~
            the name of the current frame. The current frame is called %1.")
          (cons $interpreterFrameName nil))
        (throwKeyedMsg)
          (format nil "The current frame is the only active one. Issue ~
            \)clear all to clear its contents."
          nil))
      nil))
    (progn)
      (if (null name)
        (setq $interpreterFrameRing (cdr $interpreterFrameRing))
        (progn)
          (setq found nil)
          (setq ifr nil)
          (dolist (f $interpreterFrameRing)
            (if (or found (not (equal name (frameName f))))
              (setq ifr (cons f ifr)))
            (setq found t))
          (if (null found)
            (throwKeyedMsg)
            "There is no frame called %1. Your command cannot be processed."
2.6 Global variables associated with the frame

**defvar $interpreterFrameRing**

All existing frames are kept in a ring held in this variable.

```
(defvar $interpreterFrameRing nil "The ring of all frames")
```

**defvar $interpreterFrameName**

The $interpreterFrameName variable, set in initializeInterpreterFrameRing to the constant initial to indicate that this is the initial (default) frame.

Frames are structures that capture all of the variables defined in a session. There can be multiple frames and the user can freely switch between them. Frames are kept in a ring data structure so you can move around the ring.

```
(defvar |$interpreterFrameName| '|initial|)
```

**defvar $InteractiveFrame**

$InteractiveFrame is the environment where the user values are stored. Any side effects of evaluation of a top-level expression are stored in this environment. It is always used as the starting environment for interpretation.

This variable is set in the restart function as the value returned by makeInitialModemapFrame—.
(defvar $InteractiveFrame| nil)

The $IOindex variable is the number associated with the input prompt. Every successful expression evaluated increments this number until a )clear all resets it. Here we set it to the initial value.

defvar $IOindex

— initvars —

(defvar $IOindex 1 "The current Axiom prompt number")

2.7 Interpreter Functions using Frames

The ?? function
The undoSteps function, part of the undo mechanism can reset the $InteractiveFrame.
Chapter 3

The Message Mechanism

Throughout the interpreter there are messages printed using a symbol for a database lookup. This was done to enable translation of these messages languages other than English.

Axiom messages are read from a flat file database and returned as one long string. They are preceded in the database by a key and this is how they are referenced from code. For example, one key is 2SIL0001 which means:

```
S2        Scratchpad II designation
I        from the interpreter
L        originally from LISPLIB BOOT
0001      a sequence number
```

Each message may contain formatting codes and parameter codes. The formatting codes are:

- `%ceoff` turn off centering
- `%ceon` turn on centering
- `%d` turn off bright printing
- `%f` user defined printing
- `%i` start indentation of 3 more spaces
- `%l` start a new line
- `%m` math-print an expression
- `%rjoff` turn off right justification (actually ragged left)
- `%rjon` turn on right justification (actually ragged left)
- `%s` pretty-print as an S-expression
- `%u` unindent 3 spaces
- `%x#` insert # spaces

The parameter codes look like `%1, `%2b, `%3p, `%4m, `%5bp, `%6s where the digit is the parameter number and the letters following indicate additional formatting. You can indicate as many additional formatting qualifiers as you like, to the degree they make sense.

- The “p” code means to call prefix2String on the parameter, a standard way of printing
abbreviated types.

- The “P” operator maps prefix2String over its arguments.
- The “o” operation formats the argument as an operation name.
- The “b” means to print that parameter in a bold (bright) font.
- The “c” means to center that parameter on a new line.
- The “r” means to right justify (ragged left) the argument.
- The “f” means that the parameter is a list [fn, :args] and that “fn” is to be called on “args” to get the text.

```lisp
(defun defvar (var initform &optional initvars)
  (declare (ignore initvars))
  (setf (get var 'initform) initform)
)

(defun defvar-variables (var initform &optional initvars)
  (declare (ignore initvars))
  (setf (get var 'variables) initform)
)

(defun defvar-arguments (var initform &optional initvars)
  (declare (ignore initvars))
  (setf (get var 'arguments) initform)
)

(defun defvar-variables-arguments (var initform &optional initvars)
  (declare (ignore initvars))
  (setf (get var 'variables-arguments) initform)
)

(defun defvar-arguments-variables (var initform &optional initvars)
  (declare (ignore initvars))
  (setf (get var 'arguments-variables) initform)
)

defvar $msgAlist
   — initvars —
   (defvar $msgAlist nil)

defvar $testingErrorPrefix
   — initvars —
   (defvar $testingErrorPrefix "Daly Bug")

defvar $msgdbPrims
   — initvars —
   (defvar $msgdbPrims
    (list :b "%b" :d "%d" :i "%i" :u "%u" :x "%x" :cell "%cell" :xj "%xj")
   )
defvar $msgdbPunct

— initvars —

(defvar |$msgdbPunct|
  `(|.| ! | : | ; | ? | )| "." "," ":" ";" ":?" ":?" ))

---

defvar $msgdbNoBlanksBeforeGroup

— initvars —

(defvar |$msgdbNoBlanksBeforeGroup|
  `(" " | | "%" % ,|@|$msgdbPrims| ,|@|$msgdbPunct|))

---

defvar $msgdbNoBlanksAfterGroup

— initvars —

(defvar |$msgdbNoBlanksAfterGroup|
  `(" " | | "%" % ,|@|$msgdbPrims| [ [ [ [ [ [ [ [ [ [ "" ""]))

---

defun Say a message using a keyed lookup

[sayKeyedMsgLocal p28]

— defun sayKeyedMsg —

(defun |sayKeyedMsg| (key args)
  (|sayKeyedMsgLocal| key args))

---
defun Handle msg formatting and print to file

[segmentKeyedMsg p28]
[substituteSegmentedMsg p??]
[flowSegmentedMsg p??]
[sayMSG2File p28]
[sayMSG p29]
[$printMsgsToFile p944]
[$linelength p983]
[$margin p983]
[$displayMsgNumber p950]

— defun sayKeyedMsgLocal —

(defun |sayKeyedMsgLocal| (key args)
  (let (msg msgp)
    (declare (special |$printMsgsToFile| $linelength $margin |$displayMsgNumber|))
    (setq msg (|segmentKeyedMsg| key))
    (setq msg (|substituteSegmentedMsg| msg args))
    (when |$displayMsgNumber| (setq msg `(,key |:| . ,msg)))
    (setq msgp (|flowSegmentedMsg| msg $linelength $margin))
    (when |$printMsgsToFile| (|sayMSG2File| msgp))
    (|sayMSG| msgp)))

defun Break a message into words

[string2Words p??]

— defun segmentKeyedMsg —

(defun |segmentKeyedMsg| (msg) (|string2Words| msg))

—

defun Write a msg into spadmsg.listing file

[makePathname p1193]
[deiostream p1128]
[sayBrightly1 p1200]
[shut p1128]

— defun sayMSG2File —
(defun |sayMSG2File| (msg)
  (let* (file str)
    (setq file (|makePathname| '|spadmsg| '|listing| 'a))
    (setq str (defiostream `((mode . output) (file . ,file)) 255 0))
    (sayBrightly1 msg str)
    (shut str)))

\[\text{defun sayMSG}\]

[saybrightly1 p??]
[$\text{algebraOutputStream p967}$]

— defun sayMSG —

(defun |sayMSG| (x)
  (declare (special |$\text{algebraOutputStream}$|))
  (when x (sayBrightly1 x |$\text{algebraOutputStream}$|)))

——
Chapter 4

The History Mechanism

defvar $HiFiAccess

The $HiFiAccess is set by initHist to T. It is a flag used by the history mechanism to record whether the history function is currently on. It can be reset by using the axiom command

)history off

It appears that the name means “History File Access”.
The $HiFiAccess variable is used by historySpad2Cmd to check whether history is turned on. T means it is, NIL means it is not. This is remembered in the current frame.

— initvars —

(defvar $HiFiAccess nil "Is the history function on?"

---

defvar $HistList

Thie $HistList variable is set by initHistList to an initial value of NIL elements. The last element of the list is smashed to point to the first element to make the list circular. This is a circular list of length $HistListLen. This is remembered in the current frame.

— initvars —

(defvar $HistList nil "A circular list of history elements"

---
defvar $HistListLen

The $HistListLen variable is set by initHistList to 20. This is the length of a circular list maintained in the variable $HistList. This is remembered in the current frame.

--- initvars ---
(defvar |$HistListLen| 0 "The length of the circular history list")

---

defvar $HistListAct

The $HistListAct variable is set by initHistList to 0. This variable holds the actual number of elements in the history list. This is the number of “undoable” steps. This is remembered in the current frame.

--- initvars ---
(defvar |$HistListAct| 0 "The number of undoable steps")

---

defvar $internalHistoryTable

The $internalHistoryTable variable is set at load time by a call to initvars to a value of NIL. It is part of the history mechanism. This is remembered in the current frame.

--- initvars ---
(defvar |$internalHistoryTable| nil)

---

defvar $HistRecord

The $HistRecord variable is set by initHistList to NIL. $HistRecord collects the input line, all variable bindings and the output of a step, before it is written to the file named by the function histFileName.

--- initvars ---
(defvar $HistRecord nil)

---

defvar $historyFileType

The $historyFileType is set at load time by a call to initvars to a value of "axh". It appears that this is intended to be used as a filetype extension. It is part of the history mechanism. It is used in makeHistFileName as part of the history file name.

--- initvars ---

(defvar $historyFileType nil)

---
Chapter 5

The undo mechanism

5.1 Data Structures

\$frameRecord = [delta1, delta2,... ] where delta(i) contains changes in the “backwards” direction. Each delta(i) has the form ((\text{var} . \text{proplist})...) where proplist denotes an ordinary proplist. For example, an entry of the form ((x (value) (mode (Integer)))...) indicates that to undo 1 step, x’s value is cleared and its mode should be set to (Integer).

A delta(i) of the form (systemCommand . delta) is a special delta indicating changes due to system commands executed between the last command and the current command. By recording these deltas separately, it is possible to undo to either BEFORE or AFTER the command. These special delta(i)s are given ONLY when a a system command is given which alters the environment.

recordFrame(‘system) is called before a command is executed, and recordFrame(‘normal) is called after (see processInteractive1). If no changes are found for former, no special entry is given.

The \$previousBindings is a copy of the CAAR \$InteractiveFrame. This is used to compute the delta(i)s stored in \$frameRecord.

5.2 Initial Undo Variables

\textbf{defvar} \$frameRecord

\begin{verbatim}
    — initvars —

    (defvar \$frameRecord nil “a list of value changes”)
\end{verbatim}
5.3 The undo functions

defun undo

(defun |undo| (l)
  (let (tmp1 key s undoWhen n)
    (declare (special |$options| |$InteractiveFrame|))
    (setq undoWhen '|after|)

(defvar |$previousBindings| nil "a copy of Interactive Frame info for undo")

(defvar |$reportundo| nil "t means we report the steps undo takes")

(defun |undo| (l)
  (let (tmp1 key s undoWhen n)
    (declare (special |$options| |$InteractiveFrame|))
    (setq undoWhen '|after|)

(defun |undo| (l)
  (let (tmp1 key s undoWhen n)
    (declare (special |$options| |$InteractiveFrame|))
    (setq undoWhen '|after|)

(defun |undo| (l)
  (let (tmp1 key s undoWhen n)
    (declare (special |$options| |$InteractiveFrame|))
    (setq undoWhen '|after|)

(defun |undo| (l)
  (let (tmp1 key s undoWhen n)
    (declare (special |$options| |$InteractiveFrame|))
    (setq undoWhen '|after|)

(defun |undo| (l)
  (let (tmp1 key s undoWhen n)
    (declare (special |$options| |$InteractiveFrame|))
    (setq undoWhen '|after|)

(defun |undo| (l)
  (let (tmp1 key s undoWhen n)
    (declare (special |$options| |$InteractiveFrame|))
    (setq undoWhen '|after|)

(defun |undo| (l)
  (let (tmp1 key s undoWhen n)
    (declare (special |$options| |$InteractiveFrame|))
    (setq undoWhen '|after|)

(defun |undo| (l)
  (let (tmp1 key s undoWhen n)
    (declare (special |$options| |$InteractiveFrame|))
    (setq undoWhen '|after|)

(defun |undo| (l)
  (let (tmp1 key s undoWhen n)
    (declare (special |$options| |$InteractiveFrame|))
    (setq undoWhen '|after|)

(defun |undo| (l)
  (let (tmp1 key s undoWhen n)
    (declare (special |$options| |$InteractiveFrame|))
    (setq undoWhen '|after|)

(defun |undo| (l)
  (let (tmp1 key s undoWhen n)
    (declare (special |$options| |$InteractiveFrame|))
    (setq undoWhen '|after|)

(defun |undo| (l)
  (let (tmp1 key s undoWhen n)
    (declare (special |$options| |$InteractiveFrame|))
    (setq undoWhen '|after|)

(defun |undo| (l)
  (let (tmp1 key s undoWhen n)
    (declare (special |$options| |$InteractiveFrame|))
    (setq undoWhen '|after|)

(defun |undo| (l)
  (let (tmp1 key s undoWhen n)
    (declare (special |$options| |$InteractiveFrame|))
    (setq undoWhen '|after|)

(defun |undo| (l)
  (let (tmp1 key s undoWhen n)
    (declare (special |$options| |$InteractiveFrame|))
    (setq undoWhen '|after|)

(defun |undo| (l)
  (let (tmp1 key s undoWhen n)
    (declare (special |$options| |$InteractiveFrame|))
    (setq undoWhen '|after|)

(defun |undo| (l)
  (let (tmp1 key s undoWhen n)
    (declare (special |$options| |$InteractiveFrame|))
    (setq undoWhen '|after|)

(defun |undo| (l)
  (let (tmp1 key s undoWhen n)
    (declare (special |$options| |$InteractiveFrame|))
    (setq undoWhen '|after|)
5.3. THE UNDO FUNCTIONS

(when
 (and (consp $options$)
   (eq (qcdr $options$) nil)
   (progn
     (setq tmp1 (qcar $options$))
     (and (consp tmp1)
       (eq (qcdr tmp1) nil)
       (progn (setq key (qcar tmp1)) t)))))

(cond
 ((|stringPrefix?| (setq s (pname key)) "redo")
  (setq $options$ nil)
  (|read| '((|redo.input|))))
 ((null (|stringPrefix?| s "before"))
  (|userError| "only option to undo is \"redo\""))
  (t
   (setq undoWhen '|before|))))

(if (null l)
 (setq n (- 1))
 (setq n (car l)))
(when (identp n)
  (setq n (parse-integer (pname n)))
  (unless (integerp n)
    (|userError| "undo argument must be an integer")))
 (setq |$InteractiveFrame| (|undoSteps| (|undoCount| n) undoWhen)) nil))

---

defun undoSteps

-- undoes m previous commands; if )before option, then undo one extra at end
--Example: if $IOindex now is 6 and m = 2 then general layout of $frameRecord,
-- after the call to recordFrame below will be:
-- (<change for systemcommands>
--  (<change for #5> <change for system commands>
--   (<change for #4> <change for system commands>
--    (<change for #3> <change for system commands>
--     (<change for #2> <change for system commands>
--      (<change for #1> <change for system commands>) where system
command entries are optional and identified by (systemCommand . change).
-- For a ")undo 3 )after", n = 2 and undoStep will restore the environment
-- up to, but not including <change for #3>.
-- An "undo 3 )before" will additionally restore <change for #3>.
-- Thus, the later requires one extra undo at the end.

[writeInputLines p828]
[recordFrame p1083]
CHAPTER 5. THE UNDO MECHANISM

— defun undoSteps —

(defun undoSteps (m beforeOrAfter)
  (let (tmp1 tmp2 systemDelta lastTailSeen env)
    (declare (special $IOindex $InteractiveFrame $frameRecord))
    (writeInputLines 'redo (- $IOindex m))
    (recordFrame 'normal)
    (setq env (copy (caar $InteractiveFrame)))
    (do ((i 0 (1+ i)) (framelist $frameRecord (cdr framelist)))
       ((or (> i m) (atom framelist)) nil)
      (setq env (undoSingleStep (CAR framelist) env))
    (if (and (consp framelist)
      (progn
       (setq tmp1 (qcdr framelist))
        (and (consp tmp1)
          (progn
           (setq tmp2 (qcar tmp1))
          (and (consp tmp2)
            (eq (qcar tmp2) '|systemCommand|
              (progn
                (setq systemDelta (qcdr tmp2))
                t))))))
      (setq framelist (cdr framelist))
      (setq env (undoSingleStep systemDelta env))
      (setq lastTailSeen framelist))
    (cond
t      ((eq beforeOrAfter '|before|)
       (setq env (undoSingleStep (car (cdr lastTailSeen)) env)))
      (setq $frameRecord (cdr $frameRecord))
      (setq $InteractiveFrame (list (list env))))

defun undoSingleStep

undoSingleStep(changes,env) ==
  -- Each change is a name-prop-list pair. For each change:
  -- (1) if there exists a prop-list in env, then for each prop-value change:
-- (a) if the prop exists in env, RPLAC in the change value
-- (b) otherwise, CONS it onto the front of prop-values for that name
-- (2) add change to the front of env
-- pp "----Undoing 1 step--------"
-- pp changes

[assq p1200]
[seq p??]
[exit p??]
[lassoc p??]
[undoLocalModemapHack p40]

— defun undoSingleStep —

(defun |undoSingleStep| (changes env)
  (prog (name changeList pairlist proplist prop value node)
    (return
     (seq
      (progn
       (do ((tmp0 changes (cdr tmp0)) (|change| nil))
         ((or (atom tmp0)
           (progn (setq |change| (car tmp0)) nil)
           (progn
            (setq name (car |change|))
            (setq changeList (cdr |change|))
            (setq changeList (undoLocalModemapHack changeList))
            (setq proplist (cdr pairlist))
            (setq changeList (cdr |change|))
            (setq changeList (undoLocalModemapHack changeList)))
           nil)
      nil)
     (seq
      (exit
       (progn
        (when (lassoc '|localModemap| changeList)
          (setq changeList (undoLocalModemapHack changeList)))
        (cond
         ((setq pairlist (assq name env))
          (cond
           ((setq proplist (cdr pairlist))
            (do ((tmp1 changeList (cdr tmp1)) (pair nil))
              ((or (atom tmp1)
                (progn (setq pair (car tmp1)) nil)
                (progn
                 (setq prop (car pair))
                 (setq value (cdr pair))
                 (setq value (cdr pair))
                 nil)))
             nil)
            nil)
           (seq
CHAPTER 5. THE UNDO MECHANISM

(defun undoLocalModemapHack
  (cond
   ((eq name '|localModemap|) (cons name nil))
   (t (setq tmp0 (cons
         (setq name (car pair))
         (setq value (cdr pair))
        pair))
    (setq tmp0 (nreverse0 tmp0)))

  (seq
   (exit
    (cond
      ((eq name '|localModemap|) (cons name nil))
      (t pair))
    (setq tmp0 (cons
        (setq name 'localModemap)
        (cons name nil))
      (t pair))
    (setq tmp0 (cons
        (setq name 'localModemap)
        (cons name nil)))
    (t pair))
  )
)

5.3. THE UNDO FUNCTIONS

Remove undo lines from history write

Removing undo lines from history write

| defun removeUndoLines |
| (defun removeUndoLines (u) |
| "Remove undo lines from history write" |
| (prog (xtra savedIOindex s s1 m s2 x code c n acc) |
| (declare (special $currentLine $IOindex)) |
| (return |
| (seq |
| (progn |
| (setq xtra |
| (cond |
| ((stringp $currentLine) (cons $currentLine nil)) |
| (t (reverse $currentLine)))))) |
| (setq xtra |
| (prog (tmp0) |
| (setq tmp0 nil) |
| (return |
| (do ((tmp1 xtra (cdr tmp1)) (x nil)) |
| ((or (atom tmp1) |
| (progn (setq x (car tmp1)) nil)) |
| (nreverse0 tmp0))) |
| (seq |
| (exit |
| (cond |
| ((null (|stringPrefix?| )history x)) |
| (setq tmp0 (cons x tmp0))))))))) |
| (setq u (append u xtra)) |
(cond
  ((null
      (prog (tmp2)
        (setq tmp2 nil)
        (return
          (do ((tmp3 nil tmp2) (tmp4 u (cdr tmp4)) (x nil))
            ((or tmp3 (atom tmp4) (progn (setq x (car tmp4)) nil)) tmp2)
            (seq
              (exit
                (setq tmp2
                  (or tmp2 (stringPrefix? " undo" x))))))))) u)
  (t
    (setq savedIOindex "$IOindex")
    (setq "$IOindex" 1)
    (do ((y u (cdr y)))
      ((atom y) nil)
      (seq
        (exit
          (cond
            ((eql (elt (setq x (car y)) 0) ")")
              (cond
                (((stringPrefix? "undo"
                  (setq s (trimString! x)))
                  (setq s1 (trimString! (substring s 5 nil)))
                  (cond
                    ((not (string= s1 ") redo"))
                      (setq m (charPosition! ")" s1 0))
                    (setq code
                      (cond
                        ((> (maxindex s1) m) (elt s1 (1+ m)))
                        (t ")a"))
                      (setq s2 (trimString! (substring s1 0 m)))
                      (setq n
                        (cond
                          ((string= s1 ") redo")
                            0)
                          ((not (string= s2 ""))
                            (undoCount! (parse-integer s2)))
                          (t -1))
                        (rplaca y
                          (concat ">" code (princ-to-string n)))
                        (t nil))
                      (setq acc nil)
                      (do ((y (nreverse u) (cdr y)))
                        ((atom y) nil)
                        (seq
                          (exit
                            (cond
                              ((eql (elt (setq x (car y)) 0) ")"))))
5.3. THE UNDO FUNCTIONS

(defun reportUndo
  This function is enabled by setting $reportundo to a non-nil value. An example of the output generated is:

  r := binary(22/7)

---

(1) 11.001

Type: BinaryExpansion

Properties of % ::
value was: NIL
value is: ((|BinaryExpansion|) WRAPPED . #(1 (1 1 NIL (0 0 1)))

Properties of r ::
value was: NIL
value is: ((|BinaryExpansion|) WRAPPED . #(1 (1 1 NIL (0 0 1)))

[seq p??]
[exit p??]
CHAPTER 5. THE UNDO MECHANISM

— defun reportUndo —

(defun reportUndo (acc)
  (prog (name proplist curproplist prop value)
    (declare (special $InteractiveFrame))
    (return
      (seq
        (do ((tmp0 acc (cdr tmp0)) (tmp1 nil))
            ((or (atom tmp0)
                 (progn (setq tmp1 (car tmp0)) nil)
                 (progn
                   (progn
                     (setq name (car tmp1))
                     (setq proplist (cdr tmp1))
                     tmp1)
                   nil))
         nil)
      (seq
        (exit
          (progn
            (sayBrightly!
              (concat '|Properties of | (pname name) " :"))
            (setq curproplist (lassoc name (caar $InteractiveFrame)))
            (do ((tmp2 proplist (cdr tmp2)) (tmp3 nil))
                ((or (atom tmp2)
                     (progn (setq tmp3 (car tmp2)) nil)
                     (progn
                       (progn
                         (setq prop (car tmp3))
                         (setq value (cdr tmp3))
                         tmp3)
                       nil))
                 nil)
          (seq
            (exit
              (progn
                (sayBrightlyNT
                  (cons " " (cons prop (cons " was: " nil))))
                (pp value)
                (sayBrightlyNT
                  (cons " " (cons prop (cons " is: " nil))))))
          nil)
        (progn
          nil))
      nil)
    nil))
  nil)
5.3. THE UNDO FUNCTIONS

Undo previous $n$ commands

(defun undoCount (n)
  "Undo previous $n$ commands"
  (prog (m)
    (declare (special $IOindex$))
    (return
     (progn
       (setq m
         (cond
           ((>= n 0) (- (- $IOindex$ n) 1))
           (t (- n))))
       (cond
         ((>= m $IOindex$)
          (userError (concat "Magnitude of undo argument must be less than step number (" princ-to-string $IOindex$ ").")))
         (t m)))))))
Chapter 6

Tracing

defun trace
[traceSpad2Cmd p47]

— defun trace —

(defun |trace| (1)
 (|traceSpad2Cmd| 1))

—

defun traceSpad2Cmd

[qcar p??]
[qcdr p??]
[getMapSubNames p65]
[trace1 p48]
[augmentTraceNames p65]
[traceReply p80]
[$mapSubNameAlist p??]

— defun traceSpad2Cmd —

(defun |traceSpad2Cmd| (1)
 (let (tmp1 11)
  (declare (special |$mapSubNameAlist|))
  (cond
   ((and (consp 1)
     (eq (qcar 1) '|Tuple|)
(progn
  (setq tmp1 (qcdr l))
  (and (consp tmp1)
    (eq (qcdr tmp1) nil)
    (progn
      (setq l1 (qcar tmp1))
      t)))
  (setq l l1)))
(setq $mapSubNameAlist| (getMapSubNames l)
  (|trace1| (augmentTraceNames l))
  (|traceReply|)))

---

defun trace1

[hasOption p723]
[throwKeyedMsg p??]
[unabbrev p??]
[isFunctor p??]
[getTraceOption p55]
[untraceDomainLocalOps p71]
[qslessp p1244]
[poundsign p??]
[untrace p59]
[ptimers p1052]
[say p??]
[pcounters p1053]
[selectOptionLC p751]
[resetSpacers p1051]
[resetTimers p1051]
[resetCounters p1051]
[qcar p??]
[qcdr p??]
[vecp p??]
[sayKeyedMsg p27]
[devaluate p??]
[lassoc p??]
[trace1 p48]
[delete p??]
[?t p84]
[seq p??]
[exit p??]
[transTraceItem p60]
[addassoc p??]
(defun trace1 (arg)
  (prog (|$traceNoisely| constructor ops lops templ opt a
     oldl newoptions domain tracelist optionlist domainlist
     oplist y varlist argument)
    (declare (special |$traceNoisely| |$options| |$lastUntraced| |
        |$optionAlist|)))
  (return
    (seq
      (setq |$traceNoisely| nil)
      (cond
        ((|hasOption| |$options| '|nonquietly|)
          (setq |$traceNoisely| t)))
      (cond
        ((|hasOption| |$options| '|off|)
          (cond
            ((or (setq ops (|hasOption| |$options| 'ops))
               (setq lops (|hasOption| |$options| '|local|)))
              (cond
                ((null arg)
                 (|throwKeyedMsg|
                  (format nil
                    "If you use the )off option for )trace and you also use the ~
                    )local or )ops option, you must specify the name of a ~
                    constructor. You have not done so."
                  nil))
              t
              (setq constructor
                (|unabbrev|
                (cond
                  ((atom arg) arg)
                  ((null (cdr arg))
                    (cond
                      ((atom (car arg)) (car arg))
                      (t (car (car arg)))
                      (t nil)))))
              (t nil)))
            (t (null (|isFunctor| constructor)))
            (|throwKeyedMsg|)
          (t nil)))))
(format nil
"If you use the )off option for )trace and you also use ~
the )local or )ops option, you must specify the name ~
of a constructor. What you gave after )trace is not a ~
valid constructor name."
nil))
(t
 (cond (ops (setq ops (|getTraceOption| ops)) nil))
 (cond
   (lops
    (setq lops (cdr (|getTraceOption| lops)))
    (|untraceDomainLocalOps|))
   (t nil))))))))
((and (qslessp 1 (|#| |$options|))
 (null (|hasOption| |$options| '|nonquietly|)))
 (|throwKeyedMsg|
  (format nil
   "If you use the )off option for )trace then the only other ~
only other options you can use are )nonquietly, )ops and )local.")
 nil))
(t (|untrace| arg)))))
(((hasOption| |$options| '|stats|)
  (cond
   ((qslessp 1 (|#| |$options|))
    (|throwKeyedMsg| "%1 can have no other options."
     (cons ")trace ... )stats" nil)))
  (t
   (setq temp1 (car |$options|))
   (setq opt (cdr temp1))
   (cond
    (null opt)
    (format t "-v,,,'-:@<~a~>~%" (- $linelength 2)
     " Traced function execution times ")
    (|ptimers|)
    (say " ")
    (format t "-v,,,'-:@<~a~>~%" (- $linelength 2)
     " Traced function execution counts ")
    (|pcounters|))
   (t
    (|selectOptionLC| (car opt) '(|reset|) '|optionError|)
    (|resetSpacers|)
    (|resetTimers|)
    (|resetCounters|)
    (|throwKeyedMsg|
     (format nil
      "Trace facility timers, space counts and execution counts ~
have been reset."
      nil))))))))
((setq a (|hasOption| |$options| '|restore|))
 (unless (setq oldl |$lastUntraced|)
  (t nil)))))

50

CHAPTER 6. TRACING
(setq newoptions (|=delete| a |$options|))
(if (null arg)
 (|=trace1| old1)
 (progn
 (dolist (x arg)
   (if (and (consp x)
     (progn
       (setq domain (qcar x))
       (setq oplist (qcdr x))
       t)
       (vecp domain))
       (=sayKeyedMsg| "Please retrace the domain %1."
        (cons (|=devaluate| domain) nil))
     (progn
       (setq |$options| (append newoptions (lassoc x |$optionAlist|)))
       (=trace1| (list x))))))))
((null arg) nil)
((and (consp arg) (eq (qcdr arg) nil) (eq (qcar arg) '?)) (|?t|))
(t
t (setq tracelist
 or (prog (t1)
   (setq t1 nil)
   (return
    (do ((t2 arg (cdr t2)) (x nil))
      ((or (atom t2)
        (progn (setq x (car t2)) nil))
       (nreverse0 t1))
     (seq
      (exit
       (setq t1 (cons (|=transTraceItem| x) t1)))))
    (return nil)))
  (do ((t3 tracelist (cdr t3)) (x nil))
    ((or (atom t3) (progn (setq x (car t3)) nil)) nil)
     (seq
      (exit
       (setq |$optionAlist| (addassoc x |$options| |$optionAlist|)))
     (setq optionlist (|=getTraceOptions| |$options|)))
   (setq argument
     (cond
      ((setq domainlist (lassoc '|of| optionlist))
       (cond
        ((lassoc 'ops optionlist)
        (=throwKeyedMsg| "')ops and )of cannot both be options to )trace
          nil))
      (t
       (setq oplist
       (cond
        (tracelist (list (cons 'ops tracelist))))))
(setq varlist
  (cond
    ((setq y (assoc '|vars| optionlist))
     (list (cons '|vars| y)))
    (t nil)))

(append domainlist (append oplist varlist))))

(optionlist (append tracelist optionlist))
(t tracelist))

(/TRACE,0|)
(prog (t4)
  (setq t4 nil)
  (return
   (do ((t5 argument (cdr t5)) (|funName| nil))
        ((or (atom t5)
             (progn (setq |funName| (car t5)) nil))
             (nreverse0 t4))
       (seq
        (exit
         (setq t4 (cons |funName| t4)))))))

(|saveMapSig|
(prog (t6)
  (setq t6 nil)
  (return
   (do ((t7 argument (cdr t7)) (|funName| nil))
        ((or (atom t7)
             (progn (setq |funName| (car t7)) nil))
             (nreverse0 t6))
       (seq
        (exit
         (setq t6 (cons |funName| t6))))))))))

---

defun getTraceOptions

|throwKeyedMsg p??|
|throwListOfKeyedMsgs p??|
|poundsign p??|
|seq p??|
|exit p??|
|getTraceOption p55|
|$traceErrorStack p??|

— defun getTraceOptions —

(defun |getTraceOptions| (|options|)
(prog ($traceErrorStack| optionlist temp1 key |parms|)
  (declare (special $traceErrorStack|))
  (return
    (seq
      (progn
        (setq $traceErrorStack| nil)
        (setq optionlist
          (prog (t0)
            (setq t0 nil)
            (return
              (do ((t1 |options| (cdr t1)) (x nil))
                ((or (atom t1) (progn (setq x (car t1)) nil)) (nreverse0 t0))
                (seq
                  (exit
                    (setq t0 (cons (|getTraceOption| x) t0))))))))
            (cond
              ($traceErrorStack|)
              (cond
                (null (cdr $traceErrorStack|))
                (setq temp1 (car $traceErrorStack|))
                (setq key (car temp1))
                (setq |parms| (cadr temp1))
                (|throwKeyedMsg| key (cons "" |parms|)))
              (t
                (|throwListOfKeyedMsgs|
                  "There are %i problems with your )trace system command:" (cons (1#1) $traceErrorStack|))
                (nreverse $traceErrorStack|))))
          (t optionlist))))))

----------

(defun saveMapSig
  (rassoc p??)
  (addassoc p??)
  (getMapSig p54)
  ($tracedMapSignatures p1050)
  ($mapSubNameAlist p??)
  — defun saveMapSig —

(defun |saveMapSig| (funnames)
  (let (map)
    (declare (special $tracedMapSignatures| $mapSubNameAlist|))
    (dolist (name funnames)
      (when (setq map (|rassoc| name $mapSubNameAlist|))
(setq $tracedMapSignatures|
  (addassoc name (|getMapSig| map name) $tracedMapSignatures)))

---

(defun getMapSig

[get p??]
[boot-equal p??]
[$InteractiveFrame p23]

— defun getMapSig —

(defun getMapSig (mapname subname)
  (let (lmms sig)
    (declare (special $InteractiveFrame)))
    (when (setq lmms (|get| mapname '|localModemap| $InteractiveFrame))
      (do ((t0 lmms (cdr t0)) (mm nil) (t1 nil sig))
          ((or (atom t0) (progn (setq mm (car t0)) nil) t1 nil))
       (when (boot-equal (cadr mm) subname) (setq sig (cdar mm)))))
  sig))

---

(defun getTraceOption,hn

[seq p??]
[exit p??]
[isDomainOrPackage p1061]
[stackTraceOptionError p1054]
[domainToGenvar p1054]

— defun getTraceOption,hn —

(defun getTraceOption,hn (x)
  (prog (g)
    (return
     (seq
      (if (and (atom x) (null (upper-case-p (elt (princ-to-string x) 0))))
        (exit
         (seq
          (if (isDomainOrPackage| (eval x)) (exit x))
        (exit
          (|stackTraceOptionError|)
          (cons}
(format nil
   "%1 The \(trace\) option \(of\) should be followed by the name of a "
   "\(domain\) and \%2 is not one."\)
   (cons (cons x nil) nil)))))))
(if (setq g (\(domainToGenvar\) x)) (exit g))
(exit
   (\{stackTraceOptionError\}
     (cons
      (format nil
        "%1 The \(trace\) option \(of\) should be followed by the name of a "
        "\(domain\) and \%2 is not one."\)
      (cons (cons x nil) nil)))))))

---

(defun getTraceOption

(seq p??)
   [exit p??]
   [selectOptionLC p751]
   [identp p1197]
   [stackTraceOptionError p1054]
   [concat p1197]
   [object2String p??]
   [transOnlyOption p1053]
   [qcdr p??]
   [qcar p??]
   [getTraceOption,hn p54]
   [isListOfIdentifiersOrStrings p1056]
   [isListOfIdentifiers p1055]
   [throwKeyedMsg p??]
   [$traceOptionList p1050]

--- defun getTraceOption ---

(defun |getTraceOption| (arg)
   (prog (l |opts| key a |n|)
      (declare (special |$traceOptionList|))
      (return
        (seq
          (progn
            (setq key (car arg))
            (setq l (cdr arg))
            (setq key
              (|selectOptionLC| key |$traceOptionList| '|traceOptionError|))
            (setq arg (cons key l))
            )
            )
           )
         )
       )
     )
    )
   )
  )
)
(cond
  ((member key '(|nonquietly| |timer| |nt|)) arg)
  ((eq key '|break|)
    (cond
      ((null l) (cons '|break| (cons '|before| nil)))
      (t
       (setq |opts|
         (prog (t0)
           (setq t0 nil)
           (return
            (do ((t1 l (cdr t1)) (y nil))
                ((or (atom t1)
                   (progn (setq y (car t1)) nil))
                 (nreverse0 t0))
               (seq
                (exit
                 (setq t0
                   (cons
                    (|selectOptionLC| y '|before| |after| nil) t0))))))))
    (cond
      ((prog (t2)
        (setq t2 t)
        (return
         (do ((t3 nil (null t2)) (t4 |opts| (cdr t4)) (y nil))
              ((or t3 (atom t4) (progn (setq y (car t4)) nil)) t2)
            (seq
             (exit
              (setq t2 (and t2 (identp y))))))))
      (cons '|break| |opts|))
      (t
       (|stackTraceOptionError|
        (cons
         (format nil
          "%1 The )trace option )break can only have one or both of ~
           before and after as arguments."
          (cons nil nil)))))))
  ((eq key '|restore|)
    (cond
      ((null l) arg)
      (t
       (|stackTraceOptionError|
        (cons "%1 The )trace option %2 can have no arguments."
          (cons (cons (concat "\)" (|object2String| key)) nil) nil))))))
  ((eq key '|only|) (cons '|only| (|transOnlyOption| l)))
  ((eq key '|within|)
    (cond
      ((and (consp l)
        (eq (qcdr l) nil)
        (progn (setq a (qcar l)) t)
        (identp a))

arg)
(t
  (stackTraceOptionError|
    (cons
      "%1 The \trace option \2 takes exactly one name as an argument."
      (cons (cons "\within\ nil\ nil) nil))))
((member key '(['cond| before| after|))
  (setq key
    (cond
      ((eq key '|cond|) '|when|)
      (t key)))
  (cond
    ((and (consp l)
      (eq (qcdr l) nil)
      (progn (setq a (qcar l)) t))
     (cons key l))
    (t
     (stackTraceOptionError|
      (cons
       "%1 The \trace option \2 takes exactly one expression as an argument."
       (cons (cons "\within\ nil\ nil) nil))))
  ((eq key '|depth|)
    (cond
      ((and (consp l)
        (eq (qcdr l) nil)
        (progn (setq depth (qcar l)) t)
        (integerp depth))
       arg)
      (t
       (stackTraceOptionError|
        (cons
         "%1 The \trace option \2 takes exactly one integer argument."
         (cons (cons "\within\ nil\ nil) nil))))
    ((eq key '|count|)
      (cond
        ((or (null l)
          (and (consp l)
            (eq (qcdr l) nil)
            (progn (setq count (qcar l)) t)
            (integerp count))
         arg)
        (t
         (stackTraceOptionError|
          (cons
           "%1 The \trace option \2 takes exactly one integer argument."
           (cons (cons "\within\ nil\ nil) nil))))
      ((eq key '|of|)
       (cons 'of)
       (cons 'of|)
(defun getTraceOption (hn y)
  (cond
    ((or (null y) (null (rest y))) (getTraceOption hn y))
    (t (getTraceOption hn (rest y)))))

(defvar traceOptions)

(defun setTrace (option)
  (setq traceOptions (cons option traceOptions)))

(defun unsetTrace (option)
  (setq traceOptions (remove option traceOptions)))

(defun printTrace ()
  (dolist (option traceOptions)
    (format t "Trace Option: ~s\n" option)))

(defun startTrace ()
  (setTrace :varbreak)
  (setTrace :mathprint))

(defun stopTrace ()
  (setTrace :varbreak)
  (setTrace :mathprint))

(defun toggleTrace ()
  (if (eq :varbreak (car traceOptions))
      (setTrace :mathprint)
      (setTrace :varbreak)))

(defun errorTrace (option)
  (setTrace option)
  (printTrace)
  (error "Trace Option: ~s\n" option))
defun traceOptionError

(stackTraceOptionError p1054)
(commandAmbiguityError p724)

— defun traceOptionError —

(defun |traceOptionError| (opt keys)
  (if (null keys)
    (|stackTraceOptionError|
      (cons
        "%1 Axiom does not understand the )trace option %2 which you used."
      (cons (cons opt nil) nil)))
    (|commandAmbiguityError| '|trace option| opt keys)))

———

defun genDomainTraceName

(lassoc p??)
(genvar p??)
($domainTraceNameAssoc p??)

— defun genDomainTraceName —

(defun |genDomainTraceName| (y)
  (let (u g)
    (declare (special |$domainTraceNameAssoc|))
    (if (setq u (lassoc y |$domainTraceNameAssoc|))
      u
      (progn
        (setq g (genvar))
        (setq |$domainTraceNameAssoc| (cons (cons y g) |$domainTraceNameAssoc|))))
    ))

———

defun untrace

(copy p??)
(transTraceItem p60)
(/untrace,0 p??)
(lassocSub p1057)
(removeTracedMapSigs p61)
--- defun untrace ---

(defun untrace (arg)
  (let (untracelist)
    (declare (special $lastUntraced /tracenames $mapSubNameAlist))
    (if arg
      (setq $lastUntraced arg)
      (setq $lastUntraced (copy /tracenames))
    (setq untracelist
      (do ((t1 arg (cdr t1)) (x nil) (t0 nil))
        ((or (atom t1) (progn (setq x (car t1)) nil))
         (nreverse0 t0))
      (push (|transTraceItem| x) t0))
    (/UNTRACE,0)
    (do ((t3 untracelist (cdr t3)) (funName nil) (t2 nil))
      ((or (atom t3) (progn (setq funName (car t3)) nil))
       (nreverse0 t2))
      (push (|lassocSub| funName $mapSubNameAlist) t2))
    (|removeTracedMapSigs| untracelist)))

---

defun transTraceItem

[get p??]
[member p1198]
[objMode p448]
[objVal p448]
[domainToGenvar p1054]
[unabbrev p??]
[constructor? p??]
[vecp p??]
[transTraceItem p60]
[devaluate p??]
[throwKeyedMsg p??]
[$doNotAddEmptyModeIfTrue p??]

--- defun transTraceItem ---

(defun |transTraceItem| (x)
  (prog (|$doNotAddEmptyModeIfTrue| |value| y)
    (declare (special $doNotAddEmptyModeIfTrue)))
(return
(progn
  (setq |$doNotAddEmptyModeIfTrue| t)
  (cond
    ((atom x)
      (cond
        ((and (setq |value| (|get| x '|value| |$InteractiveFrame|))
         (|member| (|objMode| |value|)
          '(((|Mode|) (|Domain|) (|SubDomain| (|Domain|)))))
         (setq x (|objVal| |value|))
         (cond
          ((setq y (|domainToGenvar| x)) y)
          (t x)))
          ((upper-case-p (elt (princ-to-string x) 0))
           (setq y (|unabbrev| x))
           (cond
            ((|constructor?| y) y)
            ((and (consp y) (|constructor?| (car y))) (car y))
            ((setq y (|domainToGenvar| x)) y)
            (t x)))
          (t x)))
        ((vecp (car x)) (|transTraceItem| (|devaluate| (car x))))
        ((setq y (|domainToGenvar| x)) y)
        (t (|throwKeyedMsg|
            "Axiom does not understand the use of %1 here." (cons x nil)))))))

---

defun removeTracedMapSigs

[$tracedMapSignatures p1050]

— defun removeTracedMapSigs —

(defun |removeTracedMapSigs| (untraceList)
  (declare (special |$tracedMapSignatures|))
  (dolist (name untraceList)
    (remprop name |$tracedMapSignatures|)))

---

defun coerceTraceArgs2E

[spadsysnamep p??]
[ pname p1195]
— defun coerceTraceArgs2E —

(defun coerceTraceArgs2E (tracename subname args)
  (declare (ignore tracename))
  (let (name)
    (declare (special |$OutputForm| |$mathTraceList| |$tracedMapSignatures|))
    (cond
      ((member (setq name subname) |$mathTraceList|)
        (if (spadsysnamep (pname name))
            (coerceSpadArgs2E (reverse (cdr (reverse args))))
            (do ((t1 '(|arg1| |arg2| |arg3| |arg4| |arg5| |arg6| |arg7| |arg8| |arg9| |arg10| |arg11| |arg12| |arg13| |arg14| |arg15| |arg16| |arg17| |arg18| |arg19|) (cdr t1))
              (name nil)
              (t2 args (cdr t2))
              (arg nil)
              (t3 (cdr (lassoc subname |$tracedMapSignatures|)) (cdr t3))
              (type nil)
              (t0 nil))
              ((or (atom t1)
                  (progn (setq name (car t1)) nil)
                  (atom t2)
                  (progn (setq arg (car t2)) nil)
                  (atom t3)
                  (progn (setq type (car t3)) nil))
                (nreverse0 t0)))
      (setq t0)
      (cons
        (list '= name
          (|objValUnwrap|
            (|coerceInteractive|
              (mkObjWrap arg type) |$OutputForm|))))
      ((spadsysnamep (pname name)) (reverse (cdr (reverse args))))
      (t args)))
  ))
defun coerceSpadArgs2E

[seq p??]
[exit p??]
[objValUnwrap p448]
[coerceInteractive p672]
[mkObjWrap p447]
[$streamCount p1006]
[$OutputForm p642]
[$tracedSpadModemap p??]

— defun coerceSpadArgs2E —

defun |coerceSpadArgs2E| (args)
(let (($streamCount 0))
 (declare (special |$streamCount| |$OutputForm| |$tracedSpadModemap|))
 (do ((t1 '(|arg1| |arg2| |arg3| |arg4| |arg5| |arg6| |arg7| |arg8|
 |arg9| |arg10| |arg11| |arg12| |arg13| |arg14| |arg15|
 |arg16| |arg17| |arg18| |arg19|) (cdr t1))
   (name nil)
   (t2 args (cdr t2))
   (arg nil)
   (t3 (cdr |$tracedSpadModemap|) (cdr t3))
   (type nil)
   (t0 nil))
     ((or (atom t1)
        (progn (setq name (car t1)) nil)
        (atom t2)
        (progn (setq arg (car t2)) nil)
        (atom t3)
        (progn (setq type (car t3)) nil))
      (nreverse0 t0))
 (seq
  (exit
   (setq t0
     (cons
      (cons '=
        (cons name
          (cons (objValUnwrap
            (coerceInteractive
              (mkObjWrap arg type)
              |$OutputForm|)) nil)))
        t0)))))

   ——
defun coerceTraceFunValue2E

(defun coerceTraceFunValue2E (tracename subname value)
  (let (name u)
    (declare (special $tracedMapSignatures $OutputForm $mathTraceList))
    (if (member (setq name subname) $mathTraceList)
      (cond
        ((spadsysnamep (pname tracename)) (coerceSpadFunValue2E value))
        ((setq u (lassoc subname $tracedMapSignatures))
         (objValUnwrap (coerceInteractive (mkObjWrap value (car u)) $OutputForm)))
        (t value)))
    value))

defun coerceSpadFunValue2E

(defun coerceSpadFunValue2E value)
  (let (streamCount)
    (declare (special streamCount tracedSpadModemap $OutputForm))
    (setq streamCount 0)
    (objValUnwrap (coerceInteractive (mkObjWrap value (car u)) $OutputForm)))
    (t value)))

— defun coerceTraceFunValue2E —

— defun coerceSpadFunValue2E —
(mkObjWrap |value| (car |$tracedSpadModemap|))
($OutputForm))))

defun getMapSubNames

[get p??]
[union p??]
[getPreviousMapSubNames p1056]
[unionq p??]
[$lastUntraced p??]
[$InteractiveFrame p23]
[/tracenames p??]

— defun getMapSubNames —

(defun |getMapSubNames| (arg)
  (let (lmm subs)
    (declare (special /tracenames |$lastUntraced| |$InteractiveFrame|))
    (setq subs nil)
    (dolist (mapname arg)
      (when (setq lmm (|get| mapname '|localModemap| |$InteractiveFrame|))
        (setq subs
          (append
            (do ((t2 lmm (cdr t2)) (t1 nil) (|mm| nil))
                 ((or (atom t2)
                     (progn (setq |mm| (CAR t2)) nil) (nreverse0 t1))
               (setq t1 (cons (cons mapname (cadr |mm|)) t1))
               subs))))
      (union| subs
        ([getPreviousMapSubNames| (unionq /tracenames |$lastUntraced|))))

— defun augmentTraceNames —

(defun |augmentTraceNames| (arg)
  (let (mml res)
    (declare (special (car |$tracedSpadModemap|))
      |$OutputForm|))))

(defun |augmentTraceNames| (arg)
  (let (mml res)
    (declare (special (car |$tracedSpadModemap|))
      |$OutputForm|))))
(declare (special |$InteractiveFrame|))
(dolist (tracename arg)
  (if (setq mml (|get| tracename |localModemap| |$InteractiveFrame|))
    (setq res
      (append
       (prog (t1)
         (setq t1 nil)
         (return
          (do ((t2 mml (cdr t2)) (|mm| nil))
            ((or (atom t2)
              (progn (setq |mm| (CAR t2)) nil))
              (nreverse0 t1))
             (setq t1 (cons (cadr |mm|) t1))))
          res))
      (setq res (cons tracename res))))
    res))

——

defun spadTrace,g

— defun spadTrace,g —

(defun |spadTrace,g| (x)
  (if (stringp x) (intern x) x))

——

defun spadTrace,isTraceable

[seq p??]
[exit p??]
[gensymp p??]
[reportSpadTrace p76]
[bpiname p??]

— defun spadTrace,isTraceable —

(defun |spadTrace,isTraceable| (x |domain|)
  (prog (n |functionSlot|)
    (return
     (seq
      (progn
       (setq n (caddr x))
      )))
    ))
x
(seq
  (if (atom (elt |domain| n)) (exit nil))
  (setq |functionSlot| (car (elt |domain| n)))
  (if (gensymp |functionSlot|)
      (exit (seq (|reportSpadTrace| '|Already Traced| x) (exit nil))))
  (if (null (bpiname |functionSlot|))
      (exit (seq
            (|reportSpadTrace| '|No function for| x)
            (exit nil))))
  (exit t)))))))

(defun spadTrace

[refvecp p??]
[aldorTrace p??]
[isDomainOrPackage p1061]
[userError p??]
[seq p??]
[exit p??]
[spadTrace,g p66]
[getOption p1068]
[removeOption p1054]
[opOf p??]
[assoc p??]
[flattenOperationAlist p1061]
[getOperationAlistFromLisplib p??]
[spadTrace,isTraceable p66]
[as-insert p??]
[bpiname p??]
[spadTrace,alias p76]
[subTypes p1055]
[constructSubst p664]
[bpitrace p??]
[rplac p??]
[printDashedLine p??]
[reportSpadTrace p76]
[setletprintflag p??]
[spadReply p1069]
[$tracedModemap p??]
[$fromSpadTrace p??]
[$letAssoc p??]
--- defun spadTrace ---

(defun spadTrace (domain options)
  (let ((tracedModemap listofoperations listofvariables
             listofbreakvars anyiftrue domainid currententry
             currentalist opstructurelist sig kind triple fn op
             mm n alias tracename sigslotnumberalist)
    (declare (special tracedModemap /tracenames $fromSpadTrace| $letAssoc|
              $reportSpadTrace| $traceNoisely|)))
    (setq $fromSpadTrace t)
    (setq tracedModemap nil)
    (cond
      ((and (consp domain)
             (refvecp (car domain))
             (eq (elt (car domain) 0) 0))
       (aldorTrace domain options))
      ((null (isDomainOrPackage domain))
       (userError "bad argument to trace"))
      (t
        (setq listofoperations
              (prog (t0)
                (setq t0 nil)
                (return
                 (do ((t1 (getOption 'ops options) (cdr t1)) (x nil))
                   ((or (atom t1) (progn (setq x (car t1)) nil)) (nreverse0 t0))
                   (seq
                    (exit
                     (setq t0 (cons (spadTrace,g x) t0))))))))
        (cond
         ((setq listofvariables (getOption 'vars options))
          (setq options (removeOption 'vars options)))
        (cond
         ((setq listofbreakvars (getOption 'varbreak options))
          (setq options (removeOption 'varbreak options)))
         (setq anyiftrue (null listofoperations))
         (setq domainid (lopOf (elt domain 0)))
         (setq currententry (assoc domain /tracenames))
         (setq currentalist (ifcdr currententry))
         (setq opstructurelist
               (flattenOperationAlist (getOperationAlistFromLisplib domainid)))
         (setq sigslotnumberalist
               (prog (t2)
                 (setq t2 nil)
                 (return
                  (do ((t3 opstructurelist (cdr t3)) (t4 nil))
                     ))))
```
((or (atom t3)
  (progn (setq t4 (CAR t3)) nil)
  (progn
    (setq op (car t4))
    (setq sig (cadr t4))
    (setq n (caddr t4))
    (setq kind (car (cddddr t4))) t4)
  nil)
  (nreverse0 t2))
(seq
  (exit
    (cond
      ((and (eq kind 'elt)
          (or anyiftrue (member op listofoperations))
          (integerp n)
          (|spadTrace,isTraceable|
            (setq triple
              (cons op (cons sig (cons n nil)))) domain))
      (setq t2 (cons triple t2))))))))
(cond
  (listofvariables
    (do ((t5 sigslotnumberalist (cdr t5)) (t6 nil))
        ((or (atom t5)
            (progn (setq t6 (car t5)) nil)
            (progn (progn (setq n (caddr t6)) t6) nil)) nil)
    (seq
      (exit
        (progn
          (setq fn (car (elt domain n)))
          (setq |$letAssoc|
            (as-insert (bpiname fn) listofvariables |$letAssoc|)))))))
(cond
  (listofbreakvars
    (do ((t7 sigslotnumberalist (cdr t7)) (t8 nil))
        ((or (atom t7)
            (progn (setq t8 (car t7)) nil)
            (progn (progn (setq n (caddr t8)) t8) nil)) nil)
    (seq
      (exit
        (progn
          (setq fn (car (elt domain n)))
          (setq |$letAssoc|
            (as-insert (bpiname fn) listofbreakvars |$letAssoc|)))))))
  (do ((t9 sigslotnumberalist (cdr t9)) (|pair| nil))
      ((or (atom t9)
          (progn (setq |pair| (car t9)) nil))
(progn
 (progn
   (setq op (car |pair|))
   (setq mm (cadr |pair|))
   (setq n (caddr |pair|))
   |pair|)
 nil))
(seq
 (exit
 (progn
   (setq alias (|spadTraceAlias| domainid op n))
   (setq $tracedModemap| (|subTypes| mm (|constructSubst| (elt domain 0))))
   (setq tracename
     (bpitrace (car (elt domain n)) alias options))
   (nconc |pair|
     (cons listofvariables
       (cons (car (elt domain n))
         (cons tracename (cons alias nil))))))
   (rplac (car (elt domain n)) tracename)))))
(setq sigslotnumberalist
 (prog (t10)
   (setq t10 nil)
   (return
     (do ((t11 sigslotnumberalist (cdr t11)) (x nil))
       ((or (atom t11) (progn (setq x (car t11)) nil)) (nreverse0 t10))
       (seq
        (exit
         (cond ((cdddr x) (setq t10 (cons x t10))))))))))
(cond
  (|$reportSpadtrace|
   (cond
    (|$traceNoisely| (|printDashedLine|))
    do ((t12 (|orderBySlotNumber| sigslotnumberalist) (cdr t12))
      (x nil))
    ((or (atom t12)
      (progn (setq x (car t12)) nil))
      nil)
    (seq (exit (|reportSpadTrace| 'tracing x))))))
(cond
  (|$letAssoc|
   (setqprintflag t))
  (cond
   (currententry
    (rplac (cdr currententry)
      (append sigslotnumberalist currentalist)))
   (t
    (setq /tracenames
      (cons (cons domain sigslotnumberalist) /tracenames))
    (|spadReply|)))))))
defun traceDomainLocalOps

[sayMSG p29]

— defun traceDomainLocalOps —

(defun |traceDomainLocalOps| ()
  (|sayMSG| '(" The )local option has been withdrawn"))
  (|sayMSG| '(" Use |ltr| to trace local functions.")))

defun untraceDomainLocalOps

[sayMSG p29]

— defun untraceDomainLocalOps —

(defun |untraceDomainLocalOps| ()
  (|sayMSG| '(" The )local option has been withdrawn"))
  (|sayMSG| '(" Use |ltr| to trace local functions.")))

defun traceDomainConstructor

[getOption p1068]
[seq p??]
[exit p??]
[spadTrace p67]
[concat p1197]
[embed p??]
[mkq p??]
[loadFunctor p1184]
[traceDomainLocalOps p71]
[$ConstructorCache p??]

— defun traceDomainConstructor —

(defun |traceDomainConstructor| (domainConstructor options)
  (prog (listOfLocalOps argl domain innerDomainConstructor)
(declare (special |$ConstructorCache|))
(return
(seq
(progn
(|loadFunctor| domainConstructor)
(setq listOfLocalOps (|getOption| 'local options))
(when listOfLocalOps (|traceDomainLocalOps|))
(cond
((and listOfLocalOps (null (|getOption| 'ops options))) nil)
(t
(do ((t2 (hget |$ConstructorCache| domainConstructor) (cdr t2))
    (t3 nil))
    ((or (atom t2)
        (progn (setq t3 (car t2)) nil)
        (progn
            (progn
                (setq argl1 (car t3))
                (setq domain (cddr t3)) t3)
            nil))
        nil)
    nil)
(seq
(exit
(|spadTrace| domain options)))))
(setq /tracenames (cons domainConstructor /tracenames))
(setq innerDomainConstructor
domainConstructor
  (intern (concat domainConstructor ";")))
(cond
((fboundp innerDomainConstructor)
  (setq domainConstructor innerDomainConstructor)))
(embed domainConstructor
  'lambda
  (cons 'rest
    (cons 'args nil))
  (cons 'prog
    (cons 'domain nil)
    (cons 'setq
      (cons 'domain
        (cons 'apply (cons domainConstructor
          (cons 'args nil))) nil)))
    (cons '|spadTrace|
      (cons 'domain
        (cons (mkq options) nil)))
    (cons (cons 'return (cons 'domain nil)) nil)))))))))))
defun untraceDomainConstructor, keepTraced?

(defun |untraceDomainConstructor, keepTraced?| (df domainConstructor)
  (prog (dc)
    (return
      (seq
        (if (and
          (and (consp df) (progn (setq dc (qcar df)) t))
            (|isDomainOrPackage| dc))
          (boot-equal (ifcar (|devaluate| dc)) domainConstructor))
            (exit (seq (|/UNTRACE,0| (cons dc nil)) (exit nil))))
          (exit t))))

---

---

defun untraceDomainConstructor

(defun |untraceDomainConstructor| (domainConstructor)
  (prog (innerDomainConstructor)
    (declare (special /tracenames))
      (untraceDomainConstructor, keepTraced? (df domainConstructor)
        (prog (dc)
          (return
            (seq
              (if (and
                (and (consp df) (progn (setq dc (qcar df)) t))
                  (|isDomainOrPackage| dc))
                (boot-equal (ifcar (|devaluate| dc)) domainConstructor))
                  (exit (seq (|/UNTRACE,0| (cons dc nil)) (exit nil))))
                  (exit t))))

---

---
CHAPTER 6. TRACING

(defun mapLetPrint
  (defun mapLetPrint (x val currentFunction)
    (setq x (getAliasIfTracedMapParameter x currentFunction))
    (setq currentFunction (getBpiNameIfTracedMap currentFunction))
    (letPrint x val currentFunction))

(defun getAliasIfTracedMapParameter
  (defun getAliasIfTracedMapParameter (p)
    (if (isSharpVarWithNum p)
        (get p)
        (exit p)
        (string2pint-n p)
        (substring p)
        nil))

---

defun mapLetPrint

[getAliasIfTracedMapParameter p74]
[getBpiNameIfTracedMap p75]
[letPrint p1062]

— defun mapLetPrint —

return
(seq
(progn
(setq /tracenames
  (prog (t0)
    (setq t0 nil)
    (return
      (do ((t1 /tracenames (cdr t1)) (df nil))
          ((or (atom t1) (progn (setq df (car t1)) nil)) (nreverse0 t0))
        (seq
          (exit
            (cond ((|untraceDomainConstructor,keepTraced?| df domainConstructor)
              (setq t0 (cons df t0)))))))))
(setq innerDomainConstructor
  (intern (concat domainConstructor ";;")))
(cond
  ((fboundp innerDomainConstructor) (unembed innerDomainConstructor))
  (t (unembed domainConstructor)))
(setq /tracenames ([delete] domainConstructor /tracenames)))))))

---

---

defun getAliasIfTracedMapParameter

[isSharpVarWithNum p1063]
[get p??]
[exit p??]
[string2pint-n p??]
[substring p256]
(defun getAliasIfTracedMapParameter (x currentFunction)
  (prog (aliasList)
    (declare (special $InteractiveFrame))
    (return
      (seq
        (cond
          ((isSharpVarWithNum x)
            (cond
              ((setq aliasList (get currentFunction 'alias $InteractiveFrame))
               (exit (elt aliasList (- (string2pint-n (substring (pname x) 1 nil) 1) 1))))))
          (t x))))))

defun getBpiNameIfTracedMap (name)
  (prog (lmm bpiName)
    (declare (special $InteractiveFrame /tracenames))
    (return
      (seq
        (cond
          ((setq lmm (get name 'localModemap $InteractiveFrame))
           (cond
             ((member (setq bpiName (cadar lmm)) /tracenames)
              (exit bpiName)))
            (t name))))))
defun spadTraceAlias

(internl p??)

— defun spadTraceAlias —

(defun spadTraceAlias (domainid op n)
  (internl domainid (intern "." "boot") op '|,| (princ-to-string n)))

———

defun reportSpadTrace

[qcar p??]
[sayBrightly p??]
[$traceNoisely p1049]

— defun reportSpadTrace —

(defun reportSpadTrace (header t0)
  (prog (op sig n t |msg| |namePart| y |tracePart|)
    (declare (special $traceNoisely!))
    (return
      (progn
        (setq op (car t0))
        (setq sig (cadr t0))
        (setq n (caddr t0))
        (setq t (cdddr t0))
        (cond
          ((null $traceNoisely!) nil)
          (t
            (setq |msg|
              (cons |header|
                (cons op
                  (cons '|'::|
                    (cons (CDR sig)
                      (cons '|' -> |
                        (cons (car sig)
                          (cons '|' in slot |
                            (cons n nil))))))))))
            (setq |namePart| nil)
            (setq |tracePart|
              (cond
                ((and (consp |t|) (progn (setq y (qcar |t|)) t) (null (null y)))
                  (cond
                    ((eq y '|all|)
                      (cons '|all| (cons '|vars| nil))))))}})
defun /tracereply

[|qcar| p??]
[|isDomainOrPackage| p1061]
[devaluate p??]
[seq p??]
[exit p??]
[/tracenames p??]

— defun /tracereply —

(defun /tracereply ()
  (prog (|d| domainlist |functionList|)
    (declare (special /tracenames))
    (return
      (seq
        (cond
          ((null /tracenames) " Nothing is traced.")
          (t
            (do ((t0 /tracenames (cdr t0)) (x nil))
                ((or (atom t0) (progn (setq x (car t0)) nil)) nil)
              (seq
                (exit
                  (cond
                    ((and (consp x)
                      (progn (setq |d| (qcar x)) t)
                        ((|isDomainOrPackage| |d|)
                          (setq domainlist (cons ((|devaluate| |d|)) domainlist))))
                        (t
                          (setq |functionList| (cons x |functionList|))))))))
            (append |functionList|
              (append domainlist (cons '|traced| nil))))))))))

defun spadUntrace

[|isDomainOrPackage| p1061]
[|userError| p??]
78

CHAPTER 6. TRACING

[getOption p1068]
[devaluate p??]
[assoc p??]
[sayMSG p29]
[bright p??]
[prefix2String p??]
[bpiname p??]
[remover p1070]
[setletprintflag p??]
[bpiuntrace p??]
[rplac p??]
[seq p??]
[exit p??]
[delasc p??]
[spadReply p1069]
[$letAssoc p??]
[/tracenames p??]
— defun spadUntrace —
(defun |spadUntrace| (domain options)
(prog (anyiftrue listofoperations domainid |pair| sigslotnumberalist
op sig n |lv| |bpiPointer| tracename alias |assocPair|
|newSigSlotNumberAlist|)
(declare (special |$letAssoc| /tracenames))
(return
(seq
(cond
((null (|isDomainOrPackage| domain))
(|userError| "bad argument to untrace"))
(t
(setq anyiftrue (null options))
(setq listofoperations (|getOption| ’|ops:| options))
(setq domainid (|devaluate| domain))
(cond
((null (setq |pair| (|assoc| domain /tracenames)))
(|sayMSG|
(cons "
No functions in"
(append
(|bright| (|prefix2String| domainid))
(cons "are now traced." nil)))))
(t
(setq sigslotnumberalist (cdr |pair|))
(do ((t0 sigslotnumberalist (cdr t0)) (|pair| nil))
((or (atom t0)
(progn (setq |pair| (car t0)) nil)
(progn
(progn


(setq op (car pair))
(setq sig (cadr pair))
(setq n (caddr pair))
(setq lv (cadddr pair))
(setq bpiPointer (car (cddddr pair)))
(setq tracename (cadr (cddddr pair)))
(setq alias (caddr (cddddr pair)))
(pair) nil)
(nil)

(seq (exit (cond ((or anyiftrue (member op listofoperations))
(progn
(bpiuntrace tracename alias)
(rplac (car (elt domain n)) bpiPointer)
(rplac (cdddr pair) nil)
(cond
((setq assocPair)
(assoc bpiname bpiPointer) $letAssoc)
(setq $letAssoc (remover $letAssoc assocPair))
(cond
((null $letAssoc) (setletprintflag nil))
(t nil))
(t nil)))))
(setq newSigSlotNumberAlist
(prog (t1)
(setq t1 nil)
(return
(do ((t2 sigslotnumberalist (cdr t2)) (x nil))
((or (atom t2) (progn (setq x (car t2)) nil)) (nreverse0 t1))
(seq (exit
(cond ((cdddr x) (setq t1 (cons x t1)))))))))
(cond
(newSigSlotNumberAlist)
(rplac (cdr pair) newSigSlotNumberAlist))
(t
(setq /tracenames (delasc domain /tracenames)
(isspaceReply))))))))

defun prTraceNames,fn
[seq p??]
[qcar p??]
--- defun prTraceNames,fn ---

(defun |prTraceNames,fn| (x)
  (prog (|d| |t|)
    (return
      (seq
        (if (and (and (consp x)
            (progn (setq |d| (qcar x)) (setq |t| (qcdr x)) t))
            (|isDomainOrPackage| |d|))
          (exit (cons (|devaluate| |d|) |t|)))
        (exit x))))

---

defun prTraceNames

[seq p??]
[exit p??]
[prTraceNames,fn p79]
[/tracenames p??]

--- defun prTraceNames ---

(defun |prTraceNames| ()
  (declare (special /tracenames))
  (seq
    (progn
      (do ((t0 /tracenames (cdr t0)) (x nil))
        ((or (atom t0) (progn (setq x (car t0)) nil)) nil)
        (seq
          (exit
            (print (|prTraceNames,fn| x))))))))

---

defun traceReply

[sayMessage p??]
[sayBrightly p??]
(defun traceReply ()
  (prog (|$domains| |$packages| |$constructors| |d| functionList|displayList|)
    (declare (special |$domains| |$packages| |$constructors| /tracenames $linelength))
    (return
      (seq
        (prog
          (setq |$domains| nil)
          (setq |$packages| nil)
          (setq |$constructors| nil)
          (cond
            ((null /tracenames) (sayMessage " Nothing is traced now."))
            (t
              (sayBrightly " ")
              (do ((t0 /tracenames (cdr t0)) (x nil))
                  ((or (atom t0) (progn (setq x (car t0)) nil)) nil)
              (seq
                (exit
                  (cond
                    ((and (consp x))
                      (progn (setq |d| (qcar x)) t) (isDomainOrPackage |d|)
                          (addTraceItem |d|))
                    nil))
            ))))
  )
)

— defun traceReply —
((atom x)  
  (cond  
    ((|isFunctor| x) (|addTraceItem| x))  
    ((isgenvar x) (|addTraceItem| (EVAL x)))  
    (t (setq |functionList| (cons x |functionList|)))))))

(setq |functionList|  
  (prog (t1)  
    (setq t1 nil)  
    (return  
      (do ((t2 |functionList| (cdr t2)) (x nil))  
          ((or (atom t2) (progn (setq x (car t2)) nil)) t1)  
        (seq  
          (exit  
            (cond  
              ((null (|isSubForRedundantMapName| x))  
                (setq t1  
                  (append t1  
                    (cons (|rassocSub| x |$mapSubNameAlist|)  
                      (cons " " nil)))))))))))

(cond  
  (|functionList|  
    (cond  
      ((eql 2 (|#| |functionList|))  
        (|sayMSG| (cons '| Function traced: | |functionList|)))  
      ((<= (+ 22 (|sayBrightlyLength| |functionList|)) $linelength)  
        (|sayMSG| (cons '| Functions traced: | |functionList|)))  
      (t  
        (|sayBrightly| " Functions traced:")  
        (|sayBrightly|  
          (|flowSegmentedMsg| |functionList| $linelength 6))))))

(cond  
  (!$domains|  
    (setq |displayList|  
      (concat  
        (prefix2String (CAR !$domains|))  
        (prog (t3)  
          (setq t3 nil)  
          (return  
            (do ((t4 (cdr !$domains|) (cdr t4)) (x nil))  
                ((or (atom t4) (progn (setq x (car t4)) nil)) t3)  
              (seq  
                (exit  
                  (setq t3  
                    (append t3 (concat ", " (prefix2String x)))))))))))

(cond  
  (atom |displayList|)  
  (setq |displayList| (cons |displayList| nil)))

(|sayBrightly| " Domains traced: ")  
(|sayBrightly| (|flowSegmentedMsg| |displayList| $linelength 6))))
(cond
  (|$packages|
    (setq |displayList|
      (concat
        (prefix2String) (car |$packages|))
      (prog (t5)
        (setq t5 nil)
        (return
          (do ((t6 (cdr |$packages|) (cdr t6)) (x nil))
            ((or (atom t6) (progn (setq x (car t6)) nil)) t5)
            (seq
              (exit
                (setq t5
                  (append t5 (concat ', (prefix2String x)))))))))))
  (cond ((atom |displayList|)
    (setq |displayList| (cons |displayList| nil))))
  (sayBrightly " Packages traced: ")
  (sayBrightly (flowSegmentedMsg |displayList| |$linelength| 6))))

(cond
  (|$constructors|
    (setq |displayList|
      (concat
        (abbreviate) (car |$constructors|))
      (prog (t7)
        (setq t7 nil)
        (return
          (do ((t8 (cdr |$constructors|) (cdr t8)) (x nil))
            ((or (atom t8) (progn (setq x (car t8)) nil)) t7)
              (seq
                (exit
                  (append t7 (concat ', (abbreviate x)))))))))))
  (cond ((atom |displayList|)
    (setq |displayList| (cons |displayList| nil))))
  (sayBrightly " Parameterized constructors traced: ")
  (sayBrightly (flowSegmentedMsg |displayList| $linelength 6)))
(t nil))))))))

defun addTraceItem
[constructor? p??]
[isDomain p??]
[devaluate p??]
[isDomainOrPackage p1061]
[$constructors p1070]
defun addTraceItem

(defun addTraceItem (d)
  (declare (special $constructors $domains $packages))
  (cond
    ((constructor? d)
      (setq $constructors (cons d $constructors)))
    ((isDomainOrPackage d)
      (setq $packages (cons (devaluate d) $packages)))))

--------------

defun ?t

[isgenvar p1064]
[get p??]
[sayMSG p29]
[bright p??]
[rassocSub p1058]
[qcar p??]
[qcdr p??]
[isDomainOrPackage p1061]
[isDomain p??]
[reportSpadTrace p76]
[take p??]
[sayBrightly p??]
[devaluate p??]
[$mapSubNameAlist p??]
[$InteractiveFrame p23]
[/tracenames p??]

--------------

defun ?t

(defun ?t ()
  (let (lm d suffix l)
    (declare (special /tracenames $InteractiveFrame $mapSubNameAlist))
    (if (null /tracenames)
      (|sayMSG| (|bright| "nothing is traced"))
      (progn
        (dolist (x /tracenames)
          (cond
            ((and (atom x) (null (isgenvar x)))
              (progn
                "...")))))
    "..."))
(cond
  ((setq llm (|get| x '|localModemap| |$InteractiveFrame|))
   (setq x (list (cadar llm)))))
(|sayMSG|
 '("Function" ,@(|bright| (|rassocSub| x |$mapSubNameAlist|))
 "traced"))))
(dolist (x /tracenames)
 (cond
   ((and (consp x)
         (progn (setq d (qcar x)) (setq l (qcdr x)) t)
         (|isDomainOrPackage| d))
    (progn
      (setq suffix "package")
      (|sayBrightly|
       ('" Functions traced in " ,suffix ,(|devaluate| d) ":"))
      (dolist (x (|orderBySlotNumber| l))
        (|reportSpadTrace| '| | (TAKE 4 x)))
      (terpri))))))))

---

defun tracelet

[gensymp p??]
[stupidIsSpadFunction p1070]
[bpname p??]
[lassoc p??]
[union p??]
[setletprintflag p??]
[isgenvar p1064]
[compileBoot p1071]
delete p??
[$traceletflag p??]
[$QuickLet p??]
[$letAssoc p??]
[$traceletFunctions p??]

— defun tracelet —

(defun |tracelet| (fn |vars|)
 (prog ($traceletflag |$QuickLet| l)
   (declare (special $traceletflag |$QuickLet| |$letAssoc|
                |$traceletFunctions|))
   (return
    (progn
      (cond

defun breaklet

[gensymp p??]
[stupidIsSpadFunction p1070]
[bpiname p??]
[lassoc p??]
[assoc p??]
[union p??]
[setletprintflag p??]
[compileBoot p1071]
[delete p??]
[$QuickLet p??]
[$letAssoc p??]
[$traceletFunctions p??]

— defun breaklet —
(defun |breaklet| (fn |vars|)
  (prog (|$QuickLet| |fnEntry| |pair|)
    (declare (special |$QuickLet| |$letAssoc| |$traceletFunctions|))
    (return
     (progn
      (cond
       ((and (gensymp fn) (|stupidIsSpadFunction| (eval fn)))
        (setq fn (eval fn))
        (cond
          ((compiled-function-p fn) (setq fn (bpiname fn)))
          (t nil))))
      (cond
       ((eq fn '|Undef|) nil)
       (t
        (setq |fnEntry| (lassoc fn |$letAssoc|))
        (setq |vars|
          (cond
           (null |fnEntry| (cons fn |$letAssoc|))
           (cons (|assoc| 'break |fnEntry|)
             (|union| |vars| (cdr |pair|)))
           (t |vars|)))
        (setq |$letAssoc|
          (cond
           (null |fnEntry|)
           (cons (cons fn (list (cons 'break |vars|))) |$letAssoc|)
           (null (|delete| fn |$traceletFunctions|))
           (null (gensymp fn))
           (progn
            (setq |$traceletFunctions| (cons fn |$traceletFunctions|))
            (|compileBoot| fn)
            (setq |$traceletFunctions|
              (|delete| fn |$traceletFunctions|))))))
    (cond
     (|$letAssoc| (setletprintflag t))
     (setq |$QuickLet| nil)
     (cond
      ((null (member fn |$traceletFunctions|))
       nil)
      (and (null (|stupidIsSpadFunction| fn))
       (null (gensymp fn)))
      (progn
       (setq |$traceletFunctions| (cons fn |$traceletFunctions|))
       (|compileBoot| fn)
       (setq |$traceletFunctions|
         (|delete| fn |$traceletFunctions|)))))))

---

defun break

[MONITOR,EVALTRAN p??]
[enable-backtrace p??]
[sayBrightly p??]
[interrupt p??]
[/breakcondition p??]
— defun break —

(defun |break| (msg)
  (prog (condition)
    (declare (special /breakcondition))
    (return
      (progn
        (setq condition (|MONITOR,EVALTRAN| /breakcondition nil))
        (when (eval condition)
          (|sayBrightly| msg)
          (interrupt))))))

——
Chapter 7

Exposure groups

7.1 Functions to manipulate exposure

Expose a group

Note that $\textit{localExposureData}$ is a vector of lists. It consists of [exposed groups, exposed constructors, hidden constructors]

\[
\begin{align*}
&\text{[object2String p?]} \\
&\text{[qcar p?] } \\
&\text{[setelt p?] } \\
&\text{[displayExposedGroups p97]} \\
&\text{[sayMSG p29]} \\
&\text{[displayExposedConstructors p97]} \\
&\text{[displayHiddenConstructors p98]} \\
&\text{[clearClams p?]} \\
&\text{[getalist p?]} \\
&\text{[sayKeyedMsg p27]} \\
&\text{[member p1198]} \\
&\text{[msort p?]} \\
&\text{[specialChar p1126]} \\
&\text{[namestring p1190]} \\
&\text{[pathname p1192]} \\
&\text{[sayAsManyPerLineAsPossible p?]} \\
&\text{[$\textit{globalExposureGroupAlist}$ p99]} \\
&\text{[$\textit{localExposureData}$ p98]} \\
&\text{[$\textit{interpreterFrameName}$ p23]} \\
&\text{[$\textit{linelength}$ p983]}
\end{align*}
\]

\textit{— defun setExposeAddGroup —}
(defun set-expose-add-group (arg)
  "Expose a group"
  (declare (special \$global-exposure-group-alist \$local-exposure-data
               \$interpreter-frame-name \$linelength))
  (if (null arg)
      (progn
        (format t "~-v,,,-:0<-a>-\%^" (- \$linelength 2) " The group Option ")
        (\display-exposed-groups)
        (\say-\msg "")
        (\say-as-many-per-line-as-possible
          (mapcar '((lambda (x) (\object2-string (first x)))
                       \$global-exposure-group-alist)))
      (dolist (x arg)
        (when (consp x) (setq x (qcar x)))
        (cond
          ((eq x '|all|) (setelt \$local-exposure-data 0
                        (mapcar '((first \$global-exposure-group-alist)))
                  (setelt \$local-exposure-data 1 nil)
                  (setelt \$local-exposure-data 2 nil)
                  (\display-exposed-groups)
                  (\say-\msg "")
                  (\display-exposed-constructors)
                  (\say-\msg "")
                  (\display-hidden-constructors)
                  (\clear-clams))
          (null (getalist \$global-exposure-group-alist x))
                  (\say-keyed-\msg "%1 is not a known exposure group name." (cons x nil)))
          ((\member x (elt \$local-exposure-data 0))
                  (\say-keyed-\msg "%1 is already an exposure group for frame %2"
                  (list x \$interpreter-frame-name)))
          (t (setelt \$local-exposure-data 0
                        (msort (cons x (elt \$local-exposure-data 0))))))
        (\say-keyed-\msg "%1 is now an exposure group for frame %2"
                  (list x \$interpreter-frame-name)))
      (\clear-clams))))

The top level set expose command handler

\[\text{displayExposedGroups p97}\]
\[\text{sayMSG p29}\]
\[\text{displayExposedConstructors p97}\]
\[\text{displayHiddenConstructors p98}\]
\[\text{sayKeyedMsg p27}\]
functions to manipulate exposure

[defun setExpose |arg|
"The top level set expose command handler"
(let (fnargs fn)
  (cond
   ((eq arg '|%initialize%|))
   ((eq arg '|%display%|) "..."
    (let (fn args)
      (cond
        ((eq fn '|add|) (|setExposeAdd| fnargs))
        ((eq fn '|drop|) (|setExposeDrop| fnargs))
        (t nil)))
   (t (|setExpose| nil)))))

The top level set expose add command handler

[specialChar p1126]
[displayExposedGroups p97]
[sayMSG p29]
[displayExposedConstructors p97]
[sayKeyedMsg p27]
[qcar p??]
[qcdr p??]
[selectOptionLC p751]
[setExposeAddGroup p89]
CHAPTER 7. EXPOSURE GROUPS

(defun setExposeAdd (arg)
  "The top level set expose add command handler"
  (declare (special $linelength))
  (let (fnargs fn)
    (cond
      ((null arg)
        (format t "\texttt{\texttt{-v,,'-\textless a\textgreater -\%}} (\textless ~\textless~ - \textgreater -\%= ~\textless~}$\text{linelength} 2) " The add Option ")
        (displayExposedGroups)
        (sayMSG "")
        (displayExposedConstructors)
        (sayMSG "")
        (sayKeyedMsg)
        (format nil
          "When \texttt{set expose add} is followed by no arguments, the information ~
          you now see is displayed. ~
          The arguments group and constructor are used to specify ~
          exposure groups or an explicit constructor to be added to the local ~
          frame exposure data. Issue ~
          \texttt{%ceon set expose add group %ceoff} or ~
          \texttt{%ceon set expose add constructor %ceoff} ~
          for more information."))
        nil))
      ((and (consp arg)
        (progn (setq fn (qcar arg)) (setq fnargs (qcdr arg)) t)
        (setq fn (selectOptionLC fn '(group |constructor|) nil)))
        (cond
          ((eq fn 'group) (setExposeAddGroup fnargs))
          ((eq fn 'constructor) (setExposeAddConstr fnargs))
          (t nil)))
      (t (setExposeAdd nil)))))

The top level set expose add constructor handler

(defun setExposeAddConstr (arg)
  "The top level set expose add constructor handler"
  (setq $linelength 2) (The add Option )
  (displayExposedGroups)
  (displayExposedConstructors)
  (sayMSG "")
  (sayKeyedMsg)
  (format nil
    "When \texttt{set expose add group} or ~
    \texttt{set expose add constructor} for more information."))
  nil))
  ((and (consp arg)
    (progn (setq fn (qcar arg)) (setq fnargs (qcdr arg)) t)
    (setq fn (selectOptionLC fn '(group |constructor|) nil)))
    (cond
      ((eq fn 'group) (setExposeAddGroup fnargs))
      ((eq fn 'constructor) (setExposeAddConstr fnargs))
      (t nil)))
    (t (setExposeAdd nil))))

---

The top level set expose add constructor handler
7.1. FUNCTIONS TO MANIPULATE EXPOSURE

(define-set-expose-add-constructor (arg)
  "The top level set expose add constructor handler"
  (declare (special $linelength $local-exposure-data $interpreter-frame-name))
  (if (null arg)
      (progn
        (print (format nil "The constructor Option")
        (display-exposed-constructors))
    (dolist (x arg)
      (let ((x (unabbrev x))
            (y (cadr x)))
        (cond
          ((null (get-database x 'constructor-kind))
            (say-keyed-msg (format nil "%s is not a known constructor. ~
                                       You can make the constructor known to the system by loading it.")
             list x))
          ((member x (elt $local-exposure-data 1))
            (say-keyed-msg "%s is already explicitly exposed in frame %s"
                           (list x $interpreter-frame-name))
          (t
           (when (member x (elt $local-exposure-data 2))
             (setelt $local-exposure-data 2
                        (cons x (elt $local-exposure-data 2))))
           (setelt $local-exposure-data 1
                    (msort (cons x (elt $local-exposure-data 1))))
           (clear-clams)
           (say-keyed-msg "%s is now explicitly exposed in frame %s"
                           (list x $interpreter-frame-name)))))))

---

The top level set expose drop handler

(define-set-expose-drop-constructor (arg)
  "The top level set expose drop constructor handler"
  (declare (special $linelength $local-exposure-data $interpreter-frame-name))
  (if (null arg)
      (progn
        (print (format nil "The constructor Option")
        (display-hidden-constructors))
    (dolist (x arg)
      (let ((x (unabbrev x))
            (y (cadr x)))
        (cond
          ((null (get-database x 'constructor-kind))
            (say-keyed-msg (format nil "%s is not a known constructor. ~
                                       You can make the constructor known to the system by loading it.")
             list x))
          ((member x (elt $local-exposure-data 1))
            (say-keyed-msg "%s is already implicitly exposed in frame %s"
                           (list x $interpreter-frame-name))
          (t
           (when (member x (elt $local-exposure-data 2))
             (setelt $local-exposure-data 2
                        (cons x (elt $local-exposure-data 2))))
           (setelt $local-exposure-data 1
                    (msort (cons x (elt $local-exposure-data 1))))
           (clear-clams)
           (say-keyed-msg "%s is now implicitly exposed in frame %s"
                           (list x $interpreter-frame-name)))))))

---
The top level set expose drop group handler
7.1. FUNCTIONS TO MANIPULATE EXPOSURE

| defun setExposeDropGroup |

(defun setExposeDropGroup (arg)
"The top level set expose drop group handler"
(declare (special $linelength |$localExposureData| |$interpreterFrameName| |$globalExposureGroupAlist|))
(if (null arg)
(progn
(format t "\n\"-%,:@<~a~>~%" (- $linelength 2) " The group Option ")
(|sayKeyedMsg|)
(format nil "When followed by one or more exposure group names, this ~
   option allows you to remove those groups from the local ~
   frame exposure data.")
nil)
(|sayMSG| " ")
(|displayExposedGroups|))
(dolist (x arg)
  (when (consp x) (setq x (qcar x)))
  (cond
   ((eq x '|all|)
    (setelt |$localExposureData| 0 nil)
    (setelt |$localExposureData| 1 nil)
    (setelt |$localExposureData| 2 nil)
    (|displayExposedGroups|)
    (|sayMSG| " ")
    (|displayExposedConstructors|)
    (|sayMSG| " ")
    (|displayHiddenConstructors|)
    (|clearClams|)))
   (((|member| x (elt |$localExposureData| 0))
    (setelt |$localExposureData| 0
     (|delete| x (elt |$localExposureData| 0)))))
    (|clearClams|)
    (|sayKeyedMsg| "%1 is no longer an exposure group for frame %2"
     (list x |$interpreterFrameName| ))))
  ((getalist |$globalExposureGroupAlist| x)
    (|sayKeyedMsg| "%1 is already an exposure group for frame %2"
     (list x |$interpreterFrameName| ))))}
The top level set expose drop constructor handler

(defun setExposeDropConstr (arg)
 "The top level set expose drop constructor handler"
 (declare (special $linelength $localExposureData $interpreterFrameName)))
 (if (null arg)
   (progn
     (format t "~*:v,,,'-:@<~a~>~%" (- $linelength 2) " The constructor Option ")
     (sayKeyedMsg)
     (format nil "When followed by one or more constructor names, this option allows you to explicitly hide constructors in this frame.")
     nil)
   (sayMSG) )
   (displayExposedConstructors)
   (sayMSG) )
   (displayHiddenConstructors))
 (dolist (x arg)
   (setq x (unabbrev x))
   (when (consp x) (setq x (qcar x)))
   (cond
     ((null (getdatabase x 'constructorkind))
      (sayKeyedMsg)
      (format nil "%1 is not a known constructor. ")
      nil)
     )
7.1. FUNCTIONS TO MANIPULATE EXPOSURE

You can make the constructor known to the system by loading it.

(list x)))

((|member| x (elt |$localExposureData| 2))
  (|sayKeyedMsg| "%1 is already explicitly hidden in frame %2"
  (list x |$interpreterFrameName|)))

(t
  (when (|member| x (elt |$localExposureData| 1))
    (setelt |$localExposureData| 1
      (|delete| x (elt |$localExposureData| 1))))
    (setelt |$localExposureData| 2
      (msort (cons x (elt |$localExposureData| 2))))
    (|clearClams|)
  (|sayKeyedMsg| "%1 is now explicitly hidden in frame %2"
    (list x |$interpreterFrameName|)))))))

---

Display exposed groups

[sayKeyedMsg p27]
[$interpreterFrameName p23]
[$localExposureData p98]

— defun displayExposedGroups —

(defun |displayExposedGroups| ()
"Display exposed groups"
  (declare (special |$interpreterFrameName| |$localExposureData|))
  (|sayKeyedMsg|)
    (format nil "The following groups are explicitly exposed in the current ~
      frame (called %1 ):")
    (list |$interpreterFrameName|))
  (if (null (elt |$localExposureData| 0))
    (format t "-v:@<~a~>~%" (- $linelength 2) " there are no exposed groups ")
    (dolist (c (elt |$localExposureData| 0))
      (format t "-v:@<~a~>~%" (- $linelength 2) c))))

---

Display exposed constructors

[sayKeyedMsg p27]
[$localExposureData p98]

— defun displayExposedConstructors —
Display hidden constructors

Display hidden constructors

7.2 Exposure Data Structures
7.2. EXPOSURE DATA STRUCTURES

defvar $localExposureDataDefault

— initvars —

(defvar $localExposureDataDefault|
  (vector |
    ;; These groups will be exposed
    (list '|basic| '|categories| '|naglink| '|anna|)
    ;; These constructors will be explicitly exposed
    (list )
    ;; These constructors will be explicitly hidden
    (list )))

NOTE: If you add new algebra you must also update this list otherwise the new algebra
won't be loaded by the interpreter when needed.

defvar $globalExposureGroupAlist

— initvars —

(defvar $globalExposureGroupAlist|
  '( |
    ;; define the groups |basic| |naglink| |anna| |categories| |Hidden| |defaults|
    (|basic|
      (|AffineAlgebraicSetComputeWithGroebnerBasis| . AFALGGR0)
      (|AffineAlgebraicSetComputeWithResultant| . AFALGRES)
      (|AffinePlane| . AFFPL)
      (|AffinePlaneOverPseudoAlgebraicClosureOfFiniteField| . AFFPLPS)
      (|AffineSpace| . AFFSP)
      (|AlgebraicManipulations| . ALGMANIP)
      (|AlgebraicNumber| . AN)
      (|AlgFactor| . ALGFACT)
      (|AlgebraicMultFact| . ALGFACT)
      (|AlgebraPackage| . ALGPKG)
      (|AlgebraGivenByStructuralConstants| . ALGSC)
      (|Any| . ANY)
      (|AnyFunctions1| . ANY1)
      (|ApplicationProgramInterface| . API)
      (|ArrayStack| . ASTACK)
      (|AssociatedJordanAlgebra| . JORDAN)
CHAPTER 7. EXPOSURE GROUPS

)|AssociatedLieAlgebra| . LIE|
(|AttachPredicates| . PMPRED|
(|AxiomServer| . AXSERV|
(|BalancedBinaryTree| . BBTREE|
(|BasicStochasticDifferential| . BSD|
(|BasicOperator| . BOP|
(|BasicOperatorFunctions1| . BOP1|
(|Bezier| . BEZIER|
(|BinaryExpansion| . BINARY|
(|BinaryFile| . BINFILE|
(|BinarySearchTree| . BSTREE|
(|BinaryTournament| . BTOURN|
(|BinaryTree| . BTREE|
(|Bits| . BITS|
(|BlasLevelOne| . BLAS1|
(|BlowUpPackage| . BLUPPACK|
(|BlowUpWithHamburgerNoether| . BLHN|
(|BlowUpWithQuadTrans| . BLQT|
(|Boolean| . BOOLEAN|
(|CardinalNumber| . CARD|
(|CartesianTensor| . CARTEN|
(|CartesianTensorFunctions2| . CARTEN2|
(|Cell| . CELL|
(|Character| . CHAR|
(|CharacterClass| . CCLASS|
(|CharacteristicPolynomialPackage| . CHARPOL|
(|CliffordAlgebra| . CLIF|
(|Color| . COLOR|
(|CommonDenominator| . CDEN|
(|Commutator| . COMM|
(|Complex| . COMPLEX|
(|ComplexDoubleFloatMatrix| . CDFMAT|
(|ComplexDoubleFloatVector| . CDFVEC|
(|ComplexFactorization| . COMPFACT|
(|ComplexFunctions2| . COMPLEX2|
(|ComplexRootPackage| . CMPLXRT|
(|ComplexTrigonometricManipulations| . CTRGMNP|
(|ContinuedFraction| . CONTFRAC|
(|CoordinateSystems| . COORDSYS|
(|CRAPackage| . CRAPACK|
(|CycleIndicators| . CYCLES|
(|CylindricalAlgebraicDecompositionPackage| . CAD|
(|CylindricalAlgebraicDecompositionUtilities| . CADU|
(|Database| . DBASE|
(|DataList| . DLIST|
(|DecimalExpansion| . DECIMAL|
(|DenavitHartenbergMatrix| . DHMATRIX|
(|Dequeue| . DEQUEUE|
(|DesingTree| . DSTREE|
(|DesingTreePackage| . DTP|
7.2. EXPOSURE DATA STRUCTURES

(|DiophantineSolutionPackage| . DIOSP)
(|DirichletRing| . DIRRING)
(|DirectProductFunctions2| . DIRPROD2)
(|DisplayPackage| . DISPLAY)
(|DistinctDegreeFactorize| . DDFACT)
(|Divisor| . DIV)
(|DoubleFloat| . DFLOT)
(|DoubleFloatMatrix| . DFMAT)
(|DoubleFloatVector| . DFVEC)
(|DoubleFloatSpecialFunctions| . DFSFUN)
(|DrawComplex| . DRAWCX)
(|DrawNumericHack| . DRAWHACK)
(|DrawOption| . DROPT)
(|EigenPackage| . EP)
(|ElementaryFunctionDefiniteIntegration| . DEFINTEF)
(|ElementaryFunctionLODESolver| . LODDEF)
(|ElementaryFunctionODESolver| . ODEEF)
(|ElementaryFunctionSign| . SIGNEF)
(|ElementaryFunctionStructurePackage| . EFSTRUC)
(|Equation| . EQ)
(|EquationFunctions2| . EQ2)
(|ErrorFunctions| . ERROR)
(|EuclideanGroebnerBasisPackage| . GBEUCLID)
(|Exit| . EXIT)
(|Export3D| . EXP3D)
(|Expression| . EXPR)
(|ExpressionFunctions2| . EXPR2)
(|ExpressionSolve| . EXPRSOL)
(|ExpressionSpaceFunctions2| . ES2)
(|ExpressionSpaceODESolver| . EXPRODE)
(|ExpressionToOpenMath| . OMEXPR)
(|ExpressionToUnivariatePowerSeries| . EXPR2UPS)
(|Factored| . FR)
(|FactoredFunctions2| . FR2)
(|FactorisationOverPseudoAlgebraicClosureOfAlgExtOfRationalNumber| . FACTEXT)
(|FactorisationOverPseudoAlgebraicClosureOfRationalNumber| . FACTRN)
(|File| . FILE)
(|FileName| . FNAME)
(|FiniteAbelianMonoidRingFunctions2| . FAMR2)
(|FiniteDivisorFunctions2| . FDIV2)
(|FiniteField| . FF)
(|FiniteFieldFactorization| . FFFACTOR)
(|FiniteFieldFactorizationWithSizeParseBySideEffect| . FFFACTSE)
(|FiniteFieldCyclicGroup| . FFCG)
(|FiniteFieldPolynomialPackage2| . FFPOLY2)
(|FiniteFieldNormalBasis| . FFBN)
(|FiniteFieldHomomorphisms| . FFHOM)
(|FiniteFieldSquareFreeDecomposition| . FFSQFR)
(|FiniteLinearAggregateFunctions2| . FLAGG2)
(|FiniteLinearAggregateSort| . FLASORT)
7.2. EXPOSURE DATA STRUCTURES

((|InfiniteTupleFunctions3| . ITFUN3)
((|InfinitelyClosePoint| . INFCLSPT)
((|InfinitelyClosePointOverPseudoAlgebraicClosureOfFiniteField| . INFCLSPS)
((|Infinity| . INFINITY)
((|Integer| . INT)
((|IntegerCombinatoricFunctions| . COMBINAT)
((|IntegerLinearDependence| . ZLINDEP)
((|IntegerNumberTheoryFunctions| . INTHEORY)
((|IntegerPrimesPackage| . PRIMES)
((|IntegerRetractions| . INTRET)
((|IntegerRoots| . IROOT)
((|IntegrationResultFunctions2| . IR2)
((|IntegrationResultRFToFunction| . IRRF2F)
((|IntegrationResultToFunction| . IR2F)
((|InterfaceGroebnerPackage| . INTERGB)
((|InterpolateFormsPackage| . INTFRSP)
((|IntersectionDivisorPackage| . INTDIVP)
((|Interval| . INTRVL)
((|InventorDataSink| . IVDATA)
((|InventorViewPort| . IVVIEW)
((|InventorRenderPackage| . IVREND)
((|InverseLaplaceTransform| . INVPLA)
((|IrrRepSymMatPackage| . IRSN)
((|KernelFunctions2| . KERNEL2)
((|KeyedAccessFile| . KAFILE)
((|LaplaceTransform| . LAPLACE)
((|LazardMorenoSolvingPackage| . LAZM3PK)
((|Library| . LIB)
((|LieSquareMatrix| . LSQM)
((|LinearOrdinaryDifferentialOperator| . LODO)
((|LinearSystemMatrixPackage| . LSMP)
((|LinearSystemMatrixPackage1| . LSMP1)
((|LinearSystemFromPowerSeriesPackage| . LISYSER)
((|LinearSystemPolynomialPackage| . LSPP)
((|List| . LIST)
((|ListOpPack| . LOP)
((|ListFunctions2| . LIST2)
((|ListFunctions3| . LIST3)
((|ListToMap| . LIST2MAP)
((|LocalParametrizationOfSimplePointPackage| . LPARSPT)
((|MakeFloatCompiledFunction| . MKFLCFN)
((|MakeFunction| . MKFUNC)
((|MakeRecord| . MKRECORD)
((|MappingPackage1| . MAPPKG1)
((|MappingPackage2| . MAPPKG2)
((|MappingPackage3| . MAPPKG3)
((|MappingPackage4| . MAPPKG4)
((|MathMLFormat| . MMLFORM)
((|Matrix| . MATRIX)
((|MatrixCategoryFunctions2| . MATCAT2)
CHAPTER 7. EXPOSURE GROUPS

(\text{MatrixCommonDenominator} . \text{MCDEN})
(\text{MatrixLinearAlgebraFunctions} . \text{MATLIN})
(\text{MatrixManipulation} . \text{MAMA})
(\text{MergeThing} . \text{MTHING})
(\text{ModularDistinctDegreeFactorizer} . \text{MDDFACT})
(\text{ModuleOperator} . \text{MODOP})
(\text{MonoidRingFunctions2} . \text{MRF2})
(\text{MoreSystemCommands} . \text{MSYSCMD})
(\text{MPolyCatFunctions2} . \text{MPC2})
(\text{MPolyCatRationalFunctionFactorizer} . \text{MPRFF})
(\text{Multiset} . \text{MSET})
(\text{MultivariateFactorize} . \text{MULTFACT})
(\text{MultivariatePolynomial} . \text{MPOLY})
(\text{MultFiniteFactorize} . \text{MFINFACT})
(\text{MyUnivariatePolynomial} . \text{MYUP})
(\text{MyExpression} . \text{MYEXPR})
(\text{NeitherSparseOrDensePowerSeries} . \text{NSDPS})
(\text{NewtonPolygon} . \text{NPOLYGON})
(\text{NoneFunctions1} . \text{NONE1})
(\text{NonNegativeInteger} . \text{NNI})
(\text{NottinghamGroup} . \text{NOTTING})
(\text{NormalizationPackage} . \text{NORMPK})
(\text{NormInMonogenicAlgebra} . \text{NORMMA})
(\text{NumberTheoreticPolynomialFunctions} . \text{NTPOLFN})
(\text{Numeric} . \text{NUMERIC})
(\text{NumericalOrdinaryDifferentialEquations} . \text{NUMODE})
(\text{NumericalQuadrature} . \text{NUMQUAD})
(\text{NumericComplexEigenPackage} . \text{NCEP})
(\text{NumericRealEigenPackage} . \text{NREP})
(\text{NumericContinuedFraction} . \text{NCNTFRAC})
(\text{Octonion} . \text{OCT})
(\text{OctonionCategoryFunctions2} . \text{OCTCT2})
(\text{OneDimensionalArray} . \text{ARRAY1})
(\text{OneDimensionalArrayFunctions2} . \text{ARRAY12})
(\text{OnePointCompletion} . \text{ONECOMP})
(\text{OnePointCompletionFunctions2} . \text{ONECOMP2})
(\text{OpenMathConnection} . \text{OMCONN})
(\text{OpenMathDevice} . \text{OMDEV})
(\text{OpenMathEncoding} . \text{OMENC})
(\text{OpenMathError} . \text{OMERR})
(\text{OpenMathErrorKind} . \text{OMERRK})
(\text{OpenMathPackage} . \text{OMPKG})
(\text{OpenMathServerPackage} . \text{OMSERVER})
(\text{OperationsQuery} . \text{OPQUERY})
(\text{OrderedCompletion} . \text{ORDCOMP})
(\text{OrderedCompletionFunctions2} . \text{ORDCOMP2})
(\text{OrdinaryDifferentialRing} . \text{ODR})
(\text{OrdSetsInts} . \text{OSI})
(\text{OrthogonalPolynomialFunctions} . \text{ORTHPOL})
(\text{OutputPackage} . \text{OUT})
7.2. EXPOSURE DATA STRUCTURES

(|PackageForAlgebraicFunctionField| . PAFF)
(|PackageForAlgebraicFunctionFieldOverFiniteField| . PAFFFF)
(|PackageForPoly| . PFORP)
(|PadeApproximantPackage| . PADEPAC)
(|Palette| . PALETTE)
(|PartialFraction| . PFR)
(|PatternFunctions2| . PATTERN2)
(|ParametricPlaneCurve| . PARPCURV)
(|ParametricSpaceCurve| . PARSCURV)
(|ParametricSurface| . PARSURF)
(|ParametricPlaneCurveFunctions2| . PARPC2)
(|ParametricSpaceCurveFunctions2| . PARSC2)
(|ParametricSurfaceFunctions2| . PARSU2)
(|ParametrizationPackage| . PARAMP)
(|PartitionsAndPermutations| . PARTPERM)
(|PatternMatch| . PATMATCH)
(|PatternMatchAssertions| . PMASS)
(|PatternMatchResultFunctions2| . PATRES2)
(|PendantTree| . PENDTREE)
(|Permanent| . PERMAN)
(|PermutationGroupExamples| . PGE)
(|PermutationGroup| . PERMGRP)
(|Permutation| . PERM)
(|Pi| . HACKPI)
(|PiCoercions| . PICERCE)
(|Places| . PLACES)
(|PlacesOverPseudoAlgebraicClosureOfFiniteField| . PLACEPS)
(|Plcs| . PLCS)
(|PointFunctions2| . PTFUNC2)
(|PolyGroebner| . PGROEB)
(|Polynomial| . POLY)
(|PolynomialAN2Expression| . PAN2EXPR)
(|PolynomialComposition| . PCOMP)
(|PolynomialDecomposition| . PDECOMP)
(|PolynomialFunctions2| . POLY2)
(|PolynomialIdeals| . IDEAL)
(|PolynomialPackageForCurve| . PLPKCRV)
(|PolynomialToUnivariatePolynomial| . POLY2UP)
(|PositiveInteger| . PI)
(|PowerSeriesLimitPackage| . LIMITPS)
(|PrimeField| . PF)
(|PrimitiveArrayFunctions2| . PRIMARR2)
(|PrintPackage| . PRINT)
(|ProjectiveAlgebraicSetPackage| . PRJALGPK)
(|ProjectivePlane| . PROJPL)
(|ProjectivePlaneOverPseudoAlgebraicClosureOfFiniteField| . PROJPLPS)
(|ProjectiveSpace| . PROJSP)
(|PseudoAlgebraicClosureOfAlgExtOfRationalNumber| . PACEXT)
(|QuadraticForm| . QFORM)
(|QuasiComponentPackage| . QCMPACK)
CHAPTER 7. EXPOSURE GROUPS

(|Quaternion| . QUAT)
(|QuaternionCategoryFunctions2| . QUATCT2)
(|QueryEquation| . QEQUAT)
(|Queue| . QUEUE)
(|QuotientFieldCategoryFunctions2| . QFCAT2)
(|RadicalEigenPackage| . REP)
(|RadicalSolvePackage| . SOLVERAD)
(|RadixExpansion| . RADIX)
(|RadixUtilities| . RADUTIL)
(|RandomNumberSource| . RANDSRC)
(|RationalFunction| . RF)
(|RationalFunctionDefiniteIntegration| . DEFINTRF)
(|RationalFunctionFactor| . RFFACT)
(|RationalFunctionFactorizer| . RFFACTOR)
(|RationalFunctionIntegration| . INTRF)
(|RationalFunctionLimitPackage| . LIMITRF)
(|RationalFunctionSign| . SIGNRF)
(|RationalFunctionSum| . SUMRF)
(|RationalRetractions| . RATRET)
(|RealClosure| . RECLOS)
(|RealPolynomialUtilitiesPackage| . POLUTIL)
(|RealZeroPackage| . REAL0)
(|RealZeroPackageQ| . REAL0Q)
(|RecurrenceOperator| . RECOP)
(|RectangularMatrixCategoryFunctions2| . RMCAT2)
(|RegularSetDecompositionPackage| . RSDCMPK)
(|RegularTriangularSet| . REGSET)
(|RegularTriangularSetGcdPackage| . RSETGCD)
(|RepresentationPackage1| . REP1)
(|RepresentationPackage2| . REP2)
(|ResolveLatticeCompletion| . RESLATC)
(|RewriteRule| . RULE)
(|RightOpenIntervalRootCharacterization| . ROIRC)
(|RomanNumeral| . ROMAN)
(|RootsFindingPackage| . RFP)
(|Ruleset| . RULESET)
(|ScriptFormulaFormat| . FORMULA)
(|ScriptFormulaFormat1| . FORMULA1)
(|Segment| . SEG)
(|SegmentBinding| . SEGBIND)
(|SegmentBindingFunctions2| . SEGBIND2)
(|SegmentFunctions2| . SEG2)
(|Set| . SET)
(|SimpleAlgebraicExtensionAlgFactor| . SAEFACT)
(|SimpleCell| . SCELL)
(|SimplifyAlgebraicNumberConvertPackage| . SIMPAN)
(|SingleInteger| . SINT)
(|SmithNormalForm| . SMITH)
(|SparseEchelonMatrix| . SEM)
(|SparseUnivariatePolynomialExpressions| . SUPEXPR)
7.2. EXPOSURE DATA STRUCTURES

(SparseUnivariatePolynomialFunctions2 . SUP2)
(SpecialOutputPackage . SPECOUT)
(SquareFreeRegularSetDecompositionPackage . SRDCMPK)
(SquareFreeRegularTriangularSet . SREGSET)
(SquareFreeRegularTriangularSetGcdPackage . SFRGCD)
(SquareFreeQuasiComponentPackage . SFQCMPK)
(Stack . STACK)
(Stream . STREAM)
(StreamFunctions1 . STREAM1)
(StreamFunctions2 . STREAM2)
(StreamFunctions3 . STREAM3)
(StreamTensor . STNSR)
(StochasticDifferential . SD)
(String . STRING)
(SturmHabichtPackage . SHP)
(Symbol . SYMBOL)
(SymmetricGroupCombinatoricFunctions . SGCF)
(SystemSolvePackage . SYSSOLP)
(SAERationalFunctionAlgFactor . SAERFFC)
(Tableau . TABLEAU)
(TaylorSeries . TS)
(TaylorSolve . UTSSOL)
(TexFormat . TEX)
(TexFormat1 . TEX1)
(TextFile . TEXTFILE)
(ThreeDimensionalViewport . VIEW3D)
(ThreeSpace . SPACE3)
(Timer . TIMER)
(TopLevelDrawFunctions . DRAW)
(TopLevelDrawFunctionsForAlgebraicCurves . DRAWCURV)
(TopLevelDrawFunctionsForCompiledFunctions . DRAWCFUN)
(TopLevelDrawFunctionsForPoints . DRAWPT)
(TopLevelThreeSpace . TOPSP)
(TranscendentalManipulations . TRMANIP)
(TransSolvePackage . SOLVETRA)
(Tree . TREE)
(TrigonometricManipulations . TRIGNIP)
(UnivariateLaurentSeriesFunctions2 . ULS2)
(UnivariateFormalPowerSeries . UFPS)
(UnivariateFormalPowerSeriesFunctions . UFPS1)
(UnivariatePolynomial . UP)
(UnivariatePolynomialCategoryFunctions2 . UPOLYC2)
(UnivariatePolynomialCommonDenominator . UPCDEN)
(UnivariatePolynomialFunctions2 . UP2)
(UnivariatePolynomialMultiplicationPackage . UMP)
(UnivariateTaylorSeriesCZero . UTSZ)
(UnivariatePuiseuxSeriesFunctions2 . UPX2)
(UnivariateTaylorSeriesFunctions2 . UTS2)
(UniversalSegment . UNISEG)
(UniversalSegmentFunctions2 . UNISEG2)

(\textbf{7.2. EXPOSURE DATA STRUCTURES})

(SparseUnivariatePolynomialFunctions2 . SUP2)
(SpecialOutputPackage . SPECOUT)
(SquareFreeRegularSetDecompositionPackage . SRDCMPK)
(SquareFreeRegularTriangularSet . SREGSET)
(SquareFreeRegularTriangularSetGcdPackage . SFRGCD)
(SquareFreeQuasiComponentPackage . SFQCMPK)
(Stack . STACK)
(Stream . STREAM)
(StreamFunctions1 . STREAM1)
(StreamFunctions2 . STREAM2)
(StreamFunctions3 . STREAM3)
(StreamTensor . STNSR)
(StochasticDifferential . SD)
(String . STRING)
(SturmHabichtPackage . SHP)
(Symbol . SYMBOL)
(SymmetricGroupCombinatoricFunctions . SGCF)
(SystemSolvePackage . SYSSOLP)
(SAERationalFunctionAlgFactor . SAERFFC)
(Tableau . TABLEAU)
(TaylorSeries . TS)
(TaylorSolve . UTSSOL)
(TexFormat . TEX)
(TexFormat1 . TEX1)
(TextFile . TEXTFILE)
(ThreeDimensionalViewport . VIEW3D)
(ThreeSpace . SPACE3)
(Timer . TIMER)
(TopLevelDrawFunctions . DRAW)
(TopLevelDrawFunctionsForAlgebraicCurves . DRAWCURV)
(TopLevelDrawFunctionsForCompiledFunctions . DRAWCFUN)
(TopLevelDrawFunctionsForPoints . DRAWPT)
(TopLevelThreeSpace . TOPSP)
(TranscendentalManipulations . TRMANIP)
(TransSolvePackage . SOLVETRA)
(Tree . TREE)
(TrigonometricManipulations . TRIGNIP)
(UnivariateLaurentSeriesFunctions2 . ULS2)
(UnivariateFormalPowerSeries . UFPS)
(UnivariateFormalPowerSeriesFunctions . UFPS1)
(UnivariatePolynomial . UP)
(UnivariatePolynomialCategoryFunctions2 . UPOLYC2)
(UnivariatePolynomialCommonDenominator . UPCDEN)
(UnivariatePolynomialFunctions2 . UP2)
(UnivariatePolynomialMultiplicationPackage . UMP)
(UnivariateTaylorSeriesCZero . UTSZ)
(UnivariatePuiseuxSeriesFunctions2 . UPX2)
(UnivariateTaylorSeriesFunctions2 . UTS2)
(UniversalSegment . UNISEG)
(UniversalSegmentFunctions2 . UNISEG2)
7.2. EXPOSURE DATA STRUCTURES

(|FortranPackage| . FORT)
(|FortranProgramCategory| . FORTCAT)
(|FortranProgram| . FORTRAN)
(|FortranFunctionCategory| . FORTFN)
(|FortranScalarType| . FST)
(|FortranType| . FT)
(|FortranTemplate| . FTEM)
(|FortranVectorFunctionCategory| . FVFUN)
(|FortranVectorCategory| . FVC)
(|MachineComplex| . MCMPLX)
(|MachineFloat| . MFLOAT)
(|MachineInteger| . MINT)
(|MultiVariableCalculusFunctions| . MCALCFN)
(|NagDiscreteFourierTransformInterfacePackage| . NAGDIS)
(|NagEigenInterfacePackage| . NAGEIG)
(|NAGLinkSupportPackage| . NAGSP)
(|NagOptimisationInterfacePackage| . NAGOPT)
(|NagQuadratureInterfacePackage| . NAQUA)
(|NagResultChecks| . NAGRES)
(|NagSpecialFunctionsInterfacePackage| . NASPE)
(|NagPolynomialRootsPackage| . NAC02)
(|NagRootFindingPackage| . NAC05)
(|NagSeriesSummationPackage| . NAC06)
(|NagIntegrationPackage| . NAGD01)
(|NagOrdinaryDifferentialEquationsPackage| . NAGD02)
(|NagPartialDifferentialEquationsPackage| . NAGD03)
(|NagInterpolationPackage| . NAGE01)
(|NagFittingPackage| . NAGE02)
(|NagOptimisationPackage| . NAGE04)
(|NagMatrixOperationsPackage| . NAGF01)
(|NagEigenPackage| . NAGF02)
(|NagLinearEquationSolvingPackage| . NAGF04)
(|NagLapack| . NAGF07)
(|NagSpecialFunctionsPackage| . NAGS)
(|PackedHermitianSequence| . PACKED)
(|Result| . RESULT)
(|SimpleFortranProgram| . SFORT)
(|Switch| . SWITCH)
(|SymbolTable| . SYMTAB)
(|TemplateUtilities| . TEMUTL)
(|TheSymbolTable| . SYMS)
(|ThreeDimensionalMatrix| . M3D))

|anna|
(|AnnaNumericalIntegrationPackage| . INTPACK)
(|AnnaNumericalOptimizationPackage| . OPTPACK)
(|AnnaOrdinaryDifferentialEquationPackage| . ODEPACK)
(|AnnaPartialDifferentialEquationPackage| . PDEPACK)
(|AttributeButtons| . ATTRBUT)
(|BasicFunctions| . BFUNCT)
(|d01ajfAnnaType| . D01AJFA)
7.2. EXPOSURE DATA STRUCTURES

(|AffineSpaceCategory| . AFSPCAT)
(|Aggregate| . AGG)
(|Algebra| . ALGEBRA)
(|AlgebraicallyClosedField| . ACF)
(|AlgebraicallyClosedFunctionSpace| . ACFS)
(|ApproximateAttribute| . ATAPPRO)
(|ArbitraryExponentAttribute| . ATARBEX)
(|ArbitraryPrecisionAttribute| . ATARBPR)
(|ArcHyperbolicFunctionCategory| . AHYP)
(|ArcTrigonometricFunctionCategory| . ATRIG)
(|AssociationListAggregate| . ALAGG)
(|AttributeRegistry| . ATTREG)
(|BagAggregate| . BGAGG)
(|BasicType| . BASTYPE)
(|BiModule| . BMODULE)
(|BinaryRecursiveAggregate| . BRAGG)
(|BinaryTreeCategory| . BTCAT)
(|BitAggregate| . BTAGG)
(|BlowUpMethodCategory| . BLMETCT)
(|CachableSet| . CACHSET)
(|CancellationAbelianMonoid| . CABMON)
(|CanonicalAttribute| . ATCANN)
(|CanonicalClosedAttribute| . ATCANCL)
(|CanonicalUnitNormalAttribute| . ATCUNOR)
(|CentralAttribute| . ATCENRL)
(|CharacteristicNonZero| . CHARNZ)
(|CharacteristicZero| . CHARZ)
(|CoercibleTo| . KOERCE)
(|Collection| . CLAGG)
(|CombinatorialFunctionCategory| . CFCAT)
(|CombinatorialOpsCategory| . COMBOPC)
(|CommutativeRing| . COMRING)
(|CommutativeStarAttribute| . ATCS)
(|Comparable| . COMPAR)
(|ComplexCategory| . COMPCAT)
(|ConvertibleTo| . KONVERT)
(|DequeueAggregate| . DQAGG)
(|DenseTreeCategory| . DSTRCAT)
(|Dictionary| . DIAGG)
(|DictionaryOperations| . DIOPS)
(|DifferentialExtension| . DIFEXT)
(|DifferentialPolynomialCategory| . DPOLCAT)
(|DifferentialRing| . DIFRING)
(|DifferentialVariableCategory| . DVARCAT)
(|DirectProductCategory| . DIRPCAT)
(|DivisionRing| . DIVRING)
(|DivisorCategory| . DIVCAT)
(|DoublyLinkedAggregate| . DLAGG)
(|ElementaryFunctionCategory| . ELEMFUN)
(|Eltable| . ELTAB)
CHAPTER 7. EXPOSURE GROUPS

(|EltableAggregate| . ELTAGG)
(|EntireRing| . ENTIRER)
(|EuclideanDomain| . EUCDOM)
(|Evalable| . EVALAB)
(|ExpressionSpace| . ES)
(|ExtensibleLinearAggregate| . ELAGG)
(|ExtensionField| . XF)
(|Field| . FIELD)
(|FieldOfPrimeCharacteristic| . FPC)
(|Finite| . FINITE)
(|FileCategory| . FILECAT)
(|FileNameCategory| . FNCAT)
(|FiniteAbelianMonoidRing| . FAMR)
(|FiniteAggregateAttribute| . ATFINAG)
(|FiniteAlgebraicExtensionField| . FAXF)
(|FiniteDivisorCategory| . FDIVCAT)
(|FiniteFieldCategory| . FFIELDC)
(|FiniteLinearAggregate| . FLAGG)
(|FiniteRankNonAssociativeAlgebra| . FINAALG)
(|FiniteRankAlgebra| . FINRALG)
(|FiniteSetAggregate| . FSAGG)
(|FloatingPointSystem| . FPS)
(|FramedAlgebra| . FRAMALG)
(|FramedNonAssociativeAlgebra| . FRNAALG)
(|FramedNonAssociativeAlgebraFunctions2| . FRNAAF2)
(|FreeAbelianMonoidCategory| . FAMONC)
(|FreeLieAlgebra| . FLALG)
(|FreeModuleCat| . FMCAT)
(|FullyEvalableOver| . FEVALAB)
(|FullyLinearlyExplicitRingOver| . FLINEXP)
(|FullyPatternMatchable| . FPATMAB)
(|FullyRetractableTo| . FRETRCT)
(|FunctionFieldCategory| . FFCAT)
(|FunctionSpace| . FS)
(|GcdDomain| . GCDDOM)
(|GradedAlgebra| . GRALG)
(|GradedModule| . GRMND)
(|Group| . GROUP)
(|HomogeneousAggregate| . HOAGG)
(|HyperbolicFunctionCategory| . HYPCAT)
(|IndexedAggregate| . IXAGG)
(|IndexedDirectProductCategory| . IDPC)
(|InfinitelyClosePointCategory| . INFCLCT)
(|InnerEvalable| . IEEVALAB)
(|IntegerNumberSystem| . INS)
(|IntegralDomain| . INTDOM)
(|IntervalCategory| . INTCAT)
(|KeyedDictionary| . KDAGG)
(|JacobiIdentityAttribute| . ATJACID)
(|LazyRepresentativeAttribute| . ATLRI


7.2. EXPOSURE DATA STRUCTURES

(|LazyStreamAggregate| . LZSTAGG)
(|LeftAlgebra| . LALG)
(|LeftModule| . LMODULE)
(|LeftOreRing| . LORER)
(|LeftUnitaryAttribute| . ATLUNIT)
(|LieAlgebra| . LIECAT)
(|LinearAggregate| . LNAGG)
(|LinearlyExplicitRingOver| . LINEXP)
(|LinearOrdinaryDifferentialOperatorCategory| . LODOCAT)
(|LiouvillianFunctionCategory| . LFCAT)
(|ListAggregate| . LSAGG)
(|LocalPowerSeriesCategory| . LOCPOWC)
(|Logic| . LOGIC)
(|ModularAlgebraicGcdOperations| . MAGCDOC)
(|MatrixCategory| . MATCAT)
(|Module| . MODULE)
(|Monad| . MONAD)
(|MonadWithUnit| . MONADWU)
(|Monoid| . MONOID)
(|MonogenicAlgebra| . MONOGEN)
(|MonogenicLinearOperator| . MLO)
(|MultiDictionary| . MDAGG)
(|MultiplicativeValuationAttribute| . ATMULVA)
(|MultisetAggregate| . MSETAGG)
(|MultivariateTaylorSeriesCategory| . MTSCAT)
(|NonAssociativeAlgebra| . NAALG)
(|NonAssociativeRing| . NASRING)
(|NonAssociativeRng| . NARNG)
(|NormalizedTriangularSetCategory| . NTSCAT)
(|NoetherianAttribute| . ATNOTHR)
(|NullSquareAttribute| . ATNULSQ)
(|NoZeroDivisorsAttribute| . ATNZDIV)
(|Object| . OBJECT)
(|OctonionCategory| . OC)
(|OneDimensionalArrayAggregate| . A1AGG)
(|OpenMath| . OM)
(|OrderedAbelianGroup| . OAGROUP)
(|OrderedAbelianMonoid| . OAMON)
(|OrderedAbelianMonoidSup| . OAMONSup)
(|OrderedAbelianSemiGroup| . OASGP)
(|OrderedCancellationAbelianMonoid| . OCAMON)
(|OrderedFinite| . ORDFIN)
(|OrderedIntegralDomain| . OINTDOM)
(|OrderedMonoid| . ORDMON)
(|OrderedMultisetAggregate| . OMSAGG)
(|OrderedRing| . ORDRING)
(|OrderedSet| . ORDSET)
(|PAdicIntegerCategory| . PADICCT)
(|PartialDifferentialRing| . PDRING)
(|PartiallyOrderedSetAttribute| . ATPOSET)
7.2. EXPOSURE DATA STRUCTURES

(SquareFreeRegularTriangularSetCategory . SFRTCAT)
(SquareMatrixCategory . SMATCAT)
(StackAggregate . SKAGG)
(StepThrough . STEP)
(StreamAggregate . STAGG)
(StringAggregate . SRAGG)
(StringCategory . STRICAT)
(StructuralConstantsPackage . SCPKG)
(TableAggregate . TBAGG)
(ThreeSpaceCategory . SPACEC)
(TrancendentalFunctionCategory . TRANFUN)
(TriangularSetCategory . TSETCAT)
(TriangularSetCategory . TRICAT)
(TwoDimensionalArrayCategory . ARR2CAT)
(Type . TYPE)
(UnaryRecursiveAggregate . URAGG)
(UniqueFactorizationDomain . UFD)
(UnitsKnownAttribute . ATUNIKN)
(UnivariateLaurentSeriesCategory . ULSCAT)
(UnivariateLaurentSeriesConstructorCategory . ULSCCAT)
(UnivariatePolynomialCategory . UPOLYC)
(UnivariatePowerSeriesCategory . UPSCAT)
(UnivariatePuiseuxSeriesCategory . UPXSCAT)
(UnivariatePuiseuxSeriesConstructorCategory . UPXSCCA)
(UnivariateTaylorSeriesCategory . UTSCAT)
(VectorCategory . VECTCAT)
(VectorSpace . VSPACE)
(XAlgebra . XALG)
(XFreeAlgebra . XFALG)
(XPolynomialsCat . XPOLYC)
(ZeroDimensionalSolvePackage . ZDSOLVE)

(AlgebraicFunction . AF)
(AlgebraicFunctionField . ALGFF)
(AlgebraicHermiteIntegration . INTHERAL)
(AlgebraicIntegrate . INTALG)
(AlgebraicIntegration . INTAF)
(AnonymousFunction . ANOUN)
(AntiSymm . ANTISYM)
(ApplyRules . APPRULE)
(ApplyUnivariateSkewPolynomial . APPLYURE)
(ArrayStack . ASTACK)
(AssociatedEquations . ASSOCEQ)
(AssociationList . ALIST)
(Automorphism . AUTOMOR)
(BalancedFactorisation . BALFACT)
(BalancedPAdicInteger . BPADIC)
(BalancedPAdicRational . BPADICRT)
(BezoutMatrix . BEZQUT)
### 7.2. EXPOSURE DATA STRUCTURES

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FiniteFieldCyclicGroupExtensionByPolynomial</td>
<td>()</td>
</tr>
<tr>
<td>FiniteFieldExtension</td>
<td>()</td>
</tr>
<tr>
<td>FiniteFieldExtensionByPolynomial</td>
<td>()</td>
</tr>
<tr>
<td>FiniteFieldFunctions</td>
<td>()</td>
</tr>
<tr>
<td>FiniteFieldNormalBasisExtension</td>
<td>()</td>
</tr>
<tr>
<td>FiniteFieldNormalBasisExtensionByPolynomial</td>
<td>()</td>
</tr>
<tr>
<td>FiniteFieldPolynomialPackage</td>
<td>()</td>
</tr>
<tr>
<td>FiniteFieldSolveLinearPolynomialEquation</td>
<td>()</td>
</tr>
<tr>
<td>FormalFraction</td>
<td>()</td>
</tr>
<tr>
<td>FourierComponent</td>
<td>()</td>
</tr>
<tr>
<td>FractionalIdeal</td>
<td>()</td>
</tr>
<tr>
<td>FramedModule</td>
<td>()</td>
</tr>
<tr>
<td>FreeAbelianGroup</td>
<td>()</td>
</tr>
<tr>
<td>FreeAbelianMonoid</td>
<td>()</td>
</tr>
<tr>
<td>FreeGroup</td>
<td>()</td>
</tr>
<tr>
<td>FreeModule</td>
<td>()</td>
</tr>
<tr>
<td>FreeModule1</td>
<td>()</td>
</tr>
<tr>
<td>FreeMonoid</td>
<td>()</td>
</tr>
<tr>
<td>FunctionalSpecialFunction</td>
<td>()</td>
</tr>
<tr>
<td>FunctionCalled</td>
<td>()</td>
</tr>
<tr>
<td>FunctionFieldIntegralBasis</td>
<td>()</td>
</tr>
<tr>
<td>FunctionSpaceReduce</td>
<td>()</td>
</tr>
<tr>
<td>FunctionSpaceToUnivariatePowerSeries</td>
<td>()</td>
</tr>
<tr>
<td>FunctionSpaceToExponentialExpansion</td>
<td>()</td>
</tr>
<tr>
<td>GaloisGroupFactorizationUtilities</td>
<td>()</td>
</tr>
<tr>
<td>GaloisGroupFactorizer</td>
<td>()</td>
</tr>
<tr>
<td>GaloisGroupPolynomialUtilities</td>
<td>()</td>
</tr>
<tr>
<td>GeneralHenselPackage</td>
<td>()</td>
</tr>
<tr>
<td>GeneralDistributedMultivariatePolynomial</td>
<td>()</td>
</tr>
<tr>
<td>GeneralPolynomialGcdPackage</td>
<td>()</td>
</tr>
<tr>
<td>GeneralSparseTable</td>
<td>()</td>
</tr>
<tr>
<td>GenericNonAssociativeAlgebra</td>
<td>()</td>
</tr>
<tr>
<td>GenExEuclid</td>
<td>()</td>
</tr>
<tr>
<td>GeneralizedMultivariateFactorize</td>
<td>()</td>
</tr>
<tr>
<td>GeneralModulePolynomial</td>
<td>()</td>
</tr>
<tr>
<td>GeneralPolynomialSet</td>
<td>()</td>
</tr>
<tr>
<td>GeneralTriangularSet</td>
<td>()</td>
</tr>
<tr>
<td>GenUFactorize</td>
<td>()</td>
</tr>
<tr>
<td>GenusZeroIntegration</td>
<td>()</td>
</tr>
<tr>
<td>GosperSummationMethod</td>
<td>()</td>
</tr>
<tr>
<td>GraphImage</td>
<td>()</td>
</tr>
<tr>
<td>GrayCode</td>
<td>()</td>
</tr>
<tr>
<td>GroebnerInternalPackage</td>
<td>()</td>
</tr>
<tr>
<td>GroebnerSolve</td>
<td>()</td>
</tr>
<tr>
<td>GuessOptionFunctions0</td>
<td>()</td>
</tr>
<tr>
<td>HashTable</td>
<td>()</td>
</tr>
<tr>
<td>Heap</td>
<td>()</td>
</tr>
<tr>
<td>HeuGcd</td>
<td>()</td>
</tr>
</tbody>
</table>
CHAPTER 7. EXPOSURE GROUPS

(HomogeneousDistributedMultivariatePolynomial . HDMP)
(HyperellipticFiniteDivisor . HELLFDIV)
(IncrementingMaps . INCRMAPS)
(IndexedBits . IBITS)
(IndexedDirectProductAbelianGroup . IDPAG)
(IndexedDirectProductAbelianMonoid . IDPAM)
(IndexedDirectProductObject . IDPO)
(IndexedDirectProductOrderedAbelianMonoid . IDPOAM)
(IndexedDirectProductOrderedAbelianMonoidSup . IDPOAMS)
(IndexedExponents . INDE)
(IndexedFlexibleArray . IFARRAY)
(IndexedList . ILIST)
(IndexedMatrix . IMATRIX)
(IndexedOneDimensionalArray . IARRAY1)
(IndexedString . ISTRING)
(IndexedTwoDimensionalArray . IARRAY2)
(IndexedVector . IVECTOR)
(InnerAlgFactor . IALGFACT)
(InnerAlgebraicNumber . IAN)
(InnerCommonDenominator . ICDEN)
(InnerFiniteField . IFF)
(InnerFreeAbelianMonoid . IFAMON)
(InnerIndexedTwoDimensionalArray . IARRAY2)
(InnerMatrixLinearAlgebraFunctions . IMATLIN)
(InnerMatrixQuotientFieldFunctions . IMATQF)
(InnerModularGcd . INMODGCD)
(InnerMultFact . INNMFACT)
(InnerNormalBasisFieldFunctions . INBFF)
(InnerNumericEigenPackage . INEP)
(InnerNumericFloatSolvePackage . INFSP)
(InnerPAdicInteger . IPADIC)
(InnerPolySign . INPSIGN)
(InnerPolySum . ISUMP)
(InnerPrimeField . IPF)
(InnerSparseUnivariatePowerSeries . ISUPS)
(InnerTable . INTABL)
(InnerTaylorSeries . ITAYLOR)
(InnerTrigonometricManipulations . ITRIGMNP)
(InputForm . INFORM)
(InputFormFunctions1 . INFORM1)
(IntegerBits . INTBIT)
(IntegerFactorizationPackage . INTFACT)
(IntegerMod . ZMOD)
(IntegerSolveLinearPolynomialEquation . INTSLPE)
(IntegralBasisPolynomialTools . IBPTOOLS)
(IntegralBasisTools . IBATOOL)
(IntegrationResult . IR)
(IntegrationTools . INTTOOLS)
(InternalPrintPackage . IPRNTPK)
(InternalRationalUnivariateRepresentationPackage . IRURPK)
7.2. EXPOSURE DATA STRUCTURES

(|IrredPolyOverFiniteField| . IRREDFFX)
(|Kernel| . KERNEL)
(|Kovacic| . KOVACIC)
(|LaurentPolynomial| . LAUPOL)
(|LeadingCoefDetermination| . LEADCDET)
(|LexTriangularPackage| . LEXTRIPK)
(|LieExponentials| . LEXP)
(|LiePolynomial| . LPOLY)
(|LinearDependence| . LINDEP)
(|LinearOrdinaryDifferentialOperatorFactorizer| . LODOF)
(|LinearOrdinaryDifferentialOperator1| . LODO1)
(|LinearOrdinaryDifferentialOperator2| . LODO2)
(|LinearOrdinaryDifferentialOperatorsOps| . LODOOPS)
(|LinearPolynomialEquationByFractions| . LPEFRAC)
(|LinGroebnerPackage| . LGROBP)
(|LiouvillianFunction| . LF)
(|ListMonoidOps| . LMOPS)
(|ListMultiDictionary| . LMDICT)
(|LocalAlgebra| . LA)
(|Localize| . LO)
(|LyndonWord| . LWORD)
(|Magma| . MAGMA)
(|MakeBinaryCompiledFunction| . MKBCF)
(|MakeCachableSet| . MKCHSET)
(|MakeUnaryCompiledFunction| . MKUCF)
(|MappingPackageInternalHacks1| . MAPHACK1)
(|MappingPackageInternalHacks2| . MAPHACK2)
(|MappingPackageInternalHacks3| . MAPHACK3)
(|MeshCreationRoutinesForThreeDimensions| . MESH)
(|ModMonic| . MODMON)
(|ModularField| . MODFIELD)
(|ModularHermitianRowReduction| . MHROWRED)
(|ModularRing| . MODRING)
(|ModuleMonomial| . MDMON)
(|MoebiusTransform| . MOEBIUS)
(|MonoidRing| . MRING)
(|MonomialExtensionTools| . MONO)
(|MPolynomialFactorizer| . MPCCF)
(|MPolynomialFunctions3| . MPCM3)
(|MRationalFactorize| . MRFAC)
(|MultipleMap| . MMAP)
(|MultivariateLifting| . MLIFT)
(|MultivariateSquareFree| . MULTSF)
(|HomogeneousDirectProduct| . HDP)
(|NewSparseMultivariatePolynomial| . NSMP)
(|NewSparseUnivariatePolynomial| . NSUP)
(|NewSparseUnivariatePolynomialFunctions2| . NSUP2)
(|NonCommutativeOperatorDivision| . NCODIV)
(|NewtonInterpolation| . NEWTON)
(|None| . NONE)
CHAPTER 7. EXPOSURE GROUPS

(|NonLinearFirstOrderODESolver| . NODE1)
(|NonLinearSolvePackage| . NLINSOL)
(|NormRetractPackage| . NORMRETR)
(|NPCoef| . NPCOEF)
(|NumberFormats| . NUMFMT)
(|NumberFieldIntegralBasis| . NFINTBAS)
(|NumericTubePlot| . NUMTUBE)
(|ODEIntegration| . ODEINT)
(|ODETools| . ODETOOLS)
(|Operator| . OP)
(|OppositeMonogenicLinearOperator| . OMLO)
(|OrderedDirectProduct| . ODP)
(|OrderedFreeMonoid| . OFMONOID)
(|OrderedVariableList| . OVAR)
(|OrderingFunctions| . ORDFUNS)
(|OrderlyDifferentialPolynomial| . ODPOL)
(|OrderlyDifferentialVariable| . ODVAR)
(|OrdinaryWeightedPolynomials| . OWP)
(|OutputForm| . OUTFORM)
(|PAdeApproximants| . PADE)
(|PAdicInteger| . PADIC)
(|PAadicRational| . PADICRAT)
(|PAadicRationalConstructor| . PADICRC)
(|PAadicWildFunctionFieldIntegralBasis| . PWFFINTB)
(|ParadoxicalCombinatorsForStreams| . YSTREAM)
(|ParametricLinearEquations| . PLEQN)
(|PartialFractionPackage| . PFRPAC)
(|Partition| . PRTITION)
(|Pattern| . PATTERN)
(|PatternFunctions1| . PATTERN1)
(|PatternMatchFunctionSpace| . PMFS)
(|PatternMatchIntegerNumberSystem| . PMINS)
(|PatternMatchIntegration| . INTPM)
(|PatternMatchKernel| . PMKERNEL)
(|PatternMatchListAggregate| . PMLSAGG)
(|PatternMatchListResult| . PATALRES)
(|PatternMatchPolynomialCategory| . PMPLCAT)
(|PatternMatchPushDown| . PMDOWN)
(|PatternMatchQuotientFieldCategory| . PMQFCAT)
(|PatternMatchResult| . PATRES)
(|PatternMatchSymbol| . PMSYM)
(|PatternMatchTools| . PMTOOLS)
(|PlaneAlgebraicCurvePlot| . ACPLOT)
(|Plot| . PLOT)
(|PlotFunctions1| . PLOT1)
(|PlotTools| . PLOTTOOL)
(|Plot3D| . PLOT3D)
(|PoincareBirkhoffWittLyndonBasis| . PBWLB)
(|Point| . POINT)
(|PointsOfFiniteOrder| . PFO)
7.2. EXPOSURE DATA STRUCTURES

(|PointsOfFiniteOrderRational| . PFOQ)
(|PointsOfFiniteOrderTools| . PFOTOOLS)
(|PointPackage| . PTPACK)
(|PolToPol| . POLTOPOL)
(|PolynomialCategoryLifting| . POLYLIFT)
(|PolynomialCategoryQuotientFunctions| . POLYCATQ)
(|PolynomialFactorizationByRecursion| . PFBR)
(|PolynomialFactorizationByRecursionUnivariate| . PFBRU)
(|PolynomialGcdPackage| . PGCD)
(|PolynomialInterpolation| . PINTERP)
(|PolynomialInterpolationAlgorithms| . PINTERPA)
(|PolynomialNumberTheoryFunctions| . PNTHEORY)
(|PolynomialRing| . PR)
(|PolynomialRoots| . POLYROOT)
(|PolynomialSetUtilitiesPackage| . PSETPK)
(|PolynomialSolveByFormulas| . SOLVEFOR)
(|PolynomialSquareFree| . PSQFR)
(|PrecomputedAssociatedEquations| . PREASSOC)
(|PrimitiveArray| . PRIMARR)
(|PrimitiveElement| . PRIMELT)
(|PrimitiveRatDE| . ODEPRIM)
(|PrimitiveRatRicDE| . ODEPRRIC)
(|Product| . PRODUCT)
(|PseudoRemainderSequence| . PRS)
(|PseudoLinearNormalForm| . PSEUDLIN)
(|PureAlgebraicIntegration| . INTPAF)
(|PureAlgebraicLODE| . ODEPAL)
(|PushVariables| . PUSHVAR)
(|QuasiAlgebraicSet| . QALGSET)
(|QuasiAlgebraicSet2| . QALGSET2)
(|RadicalFunctionField| . RADFF)
(|RandomDistributions| . RDIST)
(|RandomFloatDistributions| . RFDIST)
(|RandomIntegerDistributions| . RIDIST)
(|RationalFactorize| . RATFACT)
(|RationalIntegration| . INTRAT)
(|RationalInterpolation| . RINTERP)
(|RationalLODE| . ODERAT)
(|RationalRicDE| . ODERRIC)
(|RationalUnivariateRepresentationPackage| . RURPK)
(|RealSolvePackage| . REALSOLV)
(|RectangularMatrix| . RMATRIX)
(|ReducedDivisor| . RDIV)
(|ReduceLODE| . ODRED)
(|ReductionOfOrder| . REDORDER)
(|Reference| . REF)
(|RepeatedDoubling| . REPDB)
(|RepeatedSquaring| . REPSQ)
(|ResidueRing| . RESRING)
(|RetractSolvePackage| . RETSOL)
7.2. EXPOSURE DATA STRUCTURES

(|Tuple| . TUPLE)
(|TwoDimensionalArray| . ARRAY2)
(|TwoDimensionalPlotClipping| . CLIP)
(|TwoDimensionalViewport| . VIEW2D)
(|TwoFactorize| . TWOFACT)
(|UnivariateFactorize| . UNIFACT)
(|UnivariateLaurentSeries| . ULS)
(|UnivariateLaurentSeriesConstructor| . ULSCONS)
(|UnivariatePolynomialDecompositionPackage| . UPDECOMP)
(|UnivariatePolynomialDivisionPackage| . UPDIVP)
(|UnivariatePolynomialSquareFree| . UPSQFREE)
(|UnivariatePuiseuxSeries| . UPXS)
(|UnivariatePuiseuxSeriesConstructor| . UPXSCONS)
(|UnivariatePuiseuxSeriesWithExponentialSingularity| . UPXSSING)
(|UnivariateSkewPolynomial| . GREUP)
(|UnivariateSkewPolynomialCategoryOps| . OREPCTO)
(|UnivariateTaylorSeries| . UTS)
(|UnivariateTaylorSeriesDDESolver| . UTSODE)
(|UserDefinedPartialOrdering| . UDPO)
(|UTSodetools| . UTSODETL)
(|Variable| . VARIABLE)
(|ViewportPackage| . VIEW)
(|WeierstrassPreparation| . WEIER)
(|WeightedPolynomials| . WP)
(|WildFunctionFieldIntegralBasis| . WFFINTBS)
(|XDistributedPolynomial| . XDPOLY)
(|XExponentialPackage| . XEXPPKG)
(|XPBWPolynomial| . XPBWPOLY)
(|XPolynomial| . XPOLY)
(|XPolynomialRing| . XPR)
(|XRecursivePolynomial| . XRPOLY)
(|defaults|)
(|AbelianGroup| . ABELGRP–)
(|AbelianMonoid| . ABELMON–)
(|AbelianMonoidRing| . AMR–)
(|AbelianSemiGroup| . ABELSG–)
(|Aggregate| . AGG–)
(|Algebra| . ALGEBRA–)
(|AlgebraicallyClosedField| . ACF–)
(|AlgebraicallyClosedFunctionSpace| . ACFS–)
(|ArcTrigonometricFunctionCategory| . ATRIG–)
(|BagAggregate| . BGAGG–)
(|BasicType| . BASTYPE–)
(|BinaryRecursiveAggregate| . BRAGG–)
(|BinaryTreeCategory| . BTCAT–)
(|BitAggregate| . BTAGG–)
(|Collection| . CLAGG–)
(|ComplexCategory| . COMPCAT–)
(|Dictionary| . DIAGG–)
(|DictionaryOperations| . DIOPS–)
7.2. EXPOSURE DATA STRUCTURES

(Module| . MODULE-)
(Monad| . MONAD-)
(MonadWithUnit| . MONADWU-)
(Monoid| . MONOID-)
(MonogenicAlgebra| . MONOGEN-)
(NonAssociativeAlgebra| . NAALG-)
(NonAssociativeRing| . NASRING-)
(NonAssociativeRing| . NARNG-)
(OctonionCategory| . OC-)
(OneDimensionalArrayAggregate| . A1AGG-)
(OrderedRing| . ORDRING-)
(OrderedSet| . ORDSET-)
(PartialDifferentialRing| . PDRING-)
(PolynomialCategory| . POLYCAT-)
(PolynomialFactorizationExplicit| . PFECAT-)
(PolynomialSetCategory| . PSETCAT-)
(PowerSeriesCategory| . PSCAT-)
(QuaternionCategory| . QUATCAT-)
(QuotientFieldCategory| . QFCAT-)
(RadicalCategory| . RADCAT-)
(RealClosedField| . RCFIELD-)
(RealNumberSystem| . RNS-)
(RealRootCharacterizationCategory| . RRCC-)
(RectangularMatrixCategory| . RMATCAT-)
(RecursiveAggregate| . RCAGG-)
(RecursivePolynomialCategory| . RPOLCAT-)
(RegularTriangularSetCategory| . RSETCAT-)
(RetractableTo| . RETRACT-)
(Ring| . RING-)
(SemiGroup| . SGROUP-)
(SetAggregate| . SETAGG-)
(SetCategory| . SETCAT-)
(SquareMatrixCategory| . SMATCAT-)
(StreamAggregate| . STAGG-)
(StringAggregate| . SRAGG-)
(TableAggregate| . TBAGG-)
(TranscendentalFunctionCategory| . TRANFUN-)
(TriangularSetCategory| . TSETCAT-)
(TrigonometricFunctionCategory| . TRIGCAT-)
(TwoDimensionalArrayCategory| . ARR2CAT-)
(UnaryRecursiveAggregate| . URAGG-)
(UniqueFactorizationDomain| . UFD-)
(UnivariateLaurentSeriesConstructorCategory| . ULSCCAT-)
(UnivariatePolynomialCategory| . UPOLYC-)
(UnivariatePowerSeriesCategory| . UPSCAT-)
(UnivariatePuiseuxSeriesConstructorCategory| . UPXSCCA-)
(UnivariateSkewPolynomialCategory| . OREPCAT-)
(UnivariateTaylorSeriesCategory| . UTSCAT-)
(VectorCategory| . VECTCAT-)
(VectorSpace| . VSPACE-))
Chapter 8

The global variables

Credits

Axiom has a very long history and many people have contributed to the effort, some in large ways and some in small ways. Any and all effort deserves recognition. There is no other criteria than contribution of effort. We would like to acknowledge and thank the following people:

\begin{verbatim}
(defvar creditlist)
   — initvars —

   (defvar creditlist '("An alphabetical listing of contributors to AXIOM:")
   "Michael Albaugh Cyril Alberga Roy Adler"
   "Christian Aistleitner Richard Anderson George Andrews"
   "S.J. Atkins Henry Baker Martin Baker"
   "Stephen Balzac Yurij Baransky David R. Barton"
   "Thomas Baruchel Gerald Baumgartner Gilbert Baumsagl"
   "Michael Becker Nelson H. F. Beebe Jay Belanger"
   "David Bindel Fred Blair Vladimir Bondarenko"
   "Mark Botch Raoul Bourquin Alexandre Bouyer"
   "Karen Braman Peter A. Broadbery Martin Brock"
   "Manuel Bronstein Stephen Buchwald Florian Bundschuh"
   "Luanne Burns William Burge Ralph Byers"
   "Quentin Carpent Robert Caviness Bruce Char"
   "Ondrej Certik Tzu-Yi Chen Cheekai Chin"
   "David V. Chudnovsky Gregory V. Chudnovsky Mark Clements"
   "James Cloos Jia Zhao Cong Josh Cohen"
   "Christophe Conil Don Coppersmith George Corliss"
   "Robert Corless Gary Cornell Meino Cramer"
\end{verbatim}

127
The `$current-directory` variable is set to the current directory at startup. This is used by the `cd` function and some of the compile routines. This is the result of the (p260) get-current-directory function. This variable is used to set `*default-pathname-defaults*`. The (p264) reroot function resets it to `$spadroot`.

An example of a runtime value is:

```
$current-directory = "~/research/test/"
```

defvar $current-directory

— initvars —

`(defvar $current-directory nil)`

The `$directory-list` is a runtime list of absolute pathnames. This list is generated by (p264) reroot from the list of relative paths held in the variable `$relative-directory-list`. Each entry will be prefixed by `$spadroot`.

An example of a runtime value is:
$directory-list =
  ("/research/test/mnt/ubuntu/../../src/input/
   "/research/test/mnt/ubuntu/doc/msgs/
   "/research/test/mnt/ubuntu/../../src/algebra/
   "/research/test/mnt/ubuntu/../../src/interp/
   "/research/test/mnt/ubuntu/doc/spadhelp/")

defvar $directory-list

  — initvars —

  (defvar $directory-list nil)

The $InitialModemapFrame is used as the initial value.
See the function “makeInitialModemapFrame” (11.3 p 261).
An example of a runtime value is:
$InitialModemapFrame = '((nil))

defvar $InitialModemapFrame

  — initvars —

  (defvar $InitialModemapFrame '((nil)))

The $library-directory-list variable is the system-wide search path for library files. (p264)
reroot prepends the $spadroot variable to the $relative-library-directory-list variable.
An example of a runtime value is:
$library-directory-list = ("/research/test/mnt/ubuntu/algebra/")

defvar $library-directory-list

  — initvars —

  (defvar $library-directory-list '("/algebra/"))
The $msgDatabaseName is a locally shared variable among the message database routines. An example of a runtime value is:

|$msgDatabaseName| = nil

defvar $msgDatabaseName

— initvars —

(defvar |$msgDatabaseName| nil)

The $openServerIfTrue It appears to control whether the interpreter will be used as an open server, probably for OpenMath use. If an open server is not requested then this variable to NIL See the function “openserver” (31 p 1133). An example of a runtime value is:

$openServerIfTrue = nil

defvar $openServerIfTrue

— initvars —

(defvar $openServerIfTrue nil)

The $relative-directory-list variable contains a hand-generated list of directories used in the Axiom system. The relative directory list specifies a search path for files for the current directory structure. It has been changed from the NAG distribution back to the original form. This list is used by the (p264) reroot function to generate the absolute list of paths held in the variable $directory-list. Each entry will be prefixed by $spadroot. An example of a runtime value is:

$relative-directory-list =
The $relative-library-directory-list$ is a hand-generated list of directories containing algebra. The (p264) reroot function will prefix every path in this list with the value of the $spadroot$ variable to construct the $library-directory-list$ variable.

An example of a runtime value is:

$relative-library-directory-list = ("/algebra/")$

defvar $relative-library-directory-list$

— initvars —

(defvar $relative-library-directory-list '("/algebra/"))

The $spadroot$ variable is the internal name for the AXIOM shell variable. It is set in reroot to the value of the argument. The value is expected to be a directory name. The (p259) initroot function uses this variable if the AXIOM shell variable is not set. The (p260) make-absolute-filename function uses this path as a prefix to all of the relative filenames to make them absolute.

An example of a runtime value is:

$spadroot = "/research/test/mnt/ubuntu$"
defvar $spadroot

— initvars —

(defvar $spadroot nil)

The $SpadServer determines whether Axiom acts as a remote server. See the function “openserver” (31 p 1133).
An example of a runtime value is:

$SpadServer = nil

defvar $SpadServer

— initvars —

(defvar $SpadServer nil "t means Axiom acts as a remote server")

The $SpadServerName defines the name of the spad server socket. In unix these exist in the tmp directory as names. See the function “openserver” (31 p 1133).
An example of a runtime value is:

$SpadServerName = "/tmp/.d"

defvar $SpadServerName

— initvars —

(defvar $SpadServerName "/tmp/.d" "the name of the spad server socket")
Chapter 9

Starting Axiom

This chapter details the internal processing behind an Axiom console session where the user types “1” and gets a result.

```
axiom -nox
AXIOM Computer Algebra System
Version: Axiom (August 2014)
Timestamp: Friday September 12, 2014 at 06:24:14
----------------------------------------------------------------------------
\text{Issue } \)copyright to view copyright notices.}
\text{Issue } \)summary for a summary of useful system commands.}
\text{Issue } \)quit to leave AXIOM and return to shell.}
Visit http://axiom-developer.org for more information
----------------------------------------------------------------------------
Re-reading interp.daase
Re-reading operation.daase
Re-reading category.daase
Re-reading browse.daase
(1) ->
(1) -> 1

       (1) 1  
        Type: PositiveInteger

(2) ->

By working through this example we introduce, motivate, and explain how the interpreter works, where and why functions are called, how the system transitions from input strings to algebra, how the databases are used, and more.

If you plan to maintain or modify the interpreter this information is necessary. If you really want to know how Axiom works, this information is useful.
Each function call we describe has a link to the actual function so you can read the detailed code and see why it reacts as it does to the given input.

I’ve taken the liberty of adding comments that show the function signature. Some of the types only exist as unnamed data structures in the interpreter (e.g. "Server", which is really just a small integer). They are introduced without definition simply as a documentation aid but may sometimes be defined a Common Lisp deftypes for performance reasons.

**A Note on Common Lisp Circular Notation**

You may not be familiar with circular notation in Common Lisp. If a list contains a pointer back to itself or a sublist then the output would be an infinite stream. In order to prevent this the circular notation is used. So for a list \(X\),

```
+---|--|--|--|--
+ A | + --> + B | + --> + C | + --> + D | /
+---|--|--|--|--
```

which is the list \((A . (B . (C . (D . ()))))\). The printing rule says that if a period is followed by a parenthesis then both are suppressed. So this would print as \((A B C D)\). But it could be that we execute

```
(rplaca (last X) (cdr X))
```

so the list now is

```
+---|--|--|--|--
+ A | + --> + B | + --> + C | + --> + | /
+---|--|--|--|--
```

```
^ |
+---------------------------+
```

and now the list \(X\) is circular. This prints as

```
(A . #0=(B C #0#))
```

As you can see the \#0=\ introduces a unique label for the cons cell pointed at by (CDR A). We stored that address in the CAR of the last node. So the last node in the list uses the previously defined label with the notation \#0#.

Circular notation is used extensively in Axiom since a lot of the structures are shared or self-referential. You have to be careful because, as a result of structure sharing, changing something in one place can change an apparently unrelated structure by side-effect.

Axiom starts by invoking a function value of the lisp symbol **top-level-hook** which is normally unbound. The normal function invocation path is:

```
axiom -nox
lisp
```
9.1. AN OVERVIEW OF A SIMPLE INPUT

Here we walk through details of Axiom’s default behavior when handling a simple input, the number 1. Many details are skipped in order to provide a simple overview of the interpreter operation. Further details can be found at the specific functions.

Axiom is in `intloopReadConsole`, the Read-Eval-Print-Loop (REPL) function and the user types “1”.

SpadInterpretStream(p252) is called with a third arguments, interactive? set to t so it sets up an interactive loop to read from the console. The other two arguments are ignored on the main interpreter path.

SpadInterpretStream(p252) can also be called by the compiler, with the interactive? argument nil to read from a file. See bookvol9.

mkprompt(p267) puts one of several kinds of prompts on the screen. In the default case we include the step number. The return value is not used.

The intloopReadConsole(p254) function does tail-recursive calls to itself and never exits. It is the primary Read-Eval-Print-Loop (REPL).

intloopReadConsole(p254) reads the next line and calls one of three kinds of processors

1. intnplisp(p260) to handle )lisp input
2. ncloopCommand(p750) to handle )command input
3. intloopProcessString(p261) to handle everything else

There are only two ways out of the REPL, either using the command ”)fin” which drops into lisp or closing the *standard-input* stream. If dropped into lisp, the top level loop can be restarted by calling (restart).

intloopReadConsole takes 2 arguments. The first is a String prefix which is usually an empty string but might contain prior lines that ended with an underscore, the Axiom continuation character. The second is an Integer which will be the step number printed at the prompt.

9.1 An Overview of a Simple Input

Here we walk through details of Axiom’s default behavior when handling a simple input, the number 1. Many details are skipped in order to provide a simple overview of the interpreter operation. Further details can be found at the specific functions.
Axiom calls \texttt{serverReadLine} to read the integer from the console. First it calls \texttt{is-console} to check that the console stream exists. \texttt{sockSendInt} sends on socket 1 (\texttt{SessionManager}) a 3, meaning \texttt{EndOfOutput}, i.e. a newline. \texttt{serverSwitch} multitasks among the different sockets and finds the interpreter socket is available, returning 4 (\texttt{CallInterp}) (see sockio-c commands sent table and bookvol8).

\texttt{serverReadLine} has a cond switch for action \texttt{SCallInterp}. In that case it calls \texttt{read-line} to read the input line and returns the result, in this case, the string "1".

\texttt{intloopReadConsole} checks for various possible special kinds of input. Axiom returned a non-zero length string. Before processing it we need to check for the \"\texttt{)fin}\" condition.
9.2. Parsing the Input

We now examine the magic portion above which has several phases. The first phase constructs a data structure called a Delay. This data structure is the core data structure of the “zipper” parser.

The “zipper” parser is unique to Axiom. It was invented by Bill Burge who did research in recursive techniques, including parsing. For insight, see his article on Stream Processing Functions [Burg74].
Creating a Delay – incString

The intloopProcessString(p261) has the nested function call

```
(|intloopProcess| n t
 (|next| #'|ncloopParse|
 (|next| #'|lineoftoks| (|incString| s))))
```

which according to lisp semantics is processed inside out. First we examine the call to incString(p263) which is passed the input string “1”.

The incString(p263) function gets the string from Axiom’s input line, in this case “1” and constructs a set of nested function calls to process the input line.

```
3> (|incString| "1")
```

That result is passed to incRenumber(p298), which calls incIgen(p299) which returns a Delay(p329). It then calls incZip(p299) to “zips” together the function incRenumberLine(p300) and the two delays into a single delay. This gets put into a delay with incZip1(p299) as the function.

```
4> (|incRenumber|
   (|nonnullstream| |incLude1| 0 ("strings") (1))
5> (|Delay| |incLude1| 0 ("strings") (1))
<5 (|Delay| (|nonnullstream| |incLude1| 0 ("strings") (1)))
4> (|incIgen| (|nonnullstream| |incLude1| 0 ("strings") (1)))
<4 (|incLude| (|nonnullstream| |incIgen1| 0 ("strings") (1)))
```

```
5> (|incZip| |incRenumberLine|
   (|nonnullstream| |incLude1| 0 ("strings") (1))
   (|nonnullstream| |incIgen1| 0))
6> (|Delay| |incZip1| |incRenumberLine|)
<6 (|Delay|
    (|nonnullstream| |incZip1| |incRenumberLine|
     (|nonnullstream| |incLude1| 0 ("strings") (1))
     (|nonnullstream| |incIgen1| 0))
<5 (|incZip|
    (|nonnullstream| |incZip1| |incRenumberLine|
     (|nonnullstream| |incLude1| 0 ("strings") (1))
     (|nonnullstream| |incIgen1| 0))
```
9.2. PARSING THE INPUT

We are building a stream of functions and arguments stored in a delay structure which will eventually be evaluated. We continue this process with the call to \texttt{next} which builds a delay with the function \texttt{next1} and the current delay.

Creating a Delay – \texttt{next}

\[
\begin{align*}
\texttt{next1} & \rightarrow \texttt{next} \\texttt{lineoftoks} \\
\texttt{next} & \rightarrow \texttt{Delay} \texttt{ncloopParse} \\
\texttt{ncloopParse} & \rightarrow \texttt{next} \texttt{lineoftoks}
\end{align*}
\]

We continue building a larger delay, this time with a call to \texttt{next} with the function \texttt{ncloopParse} and the existing delay.

Creating a Delay – \texttt{ncloopParse}

\[
\begin{align*}
\texttt{ncloopParse} & \rightarrow \texttt{next} \texttt{lineoftoks} \\
\texttt{lineoftoks} & \rightarrow \texttt{nullstream} \texttt{incZip1} \texttt{incRenumberLine} \\
\texttt{nullstream} & \rightarrow \texttt{incLude1} 0 \texttt{strings} (1) \\
\texttt{incZip1} & \rightarrow \texttt{incIgen1} 0
\end{align*}
\]
Finally we call `intloopProcess` with the step number `stepno`, whether we are talking to the console `interactive` and the delay we just constructed `delay`.

### Evaluating a Delay – intloopProcess

At this point we have created a large delay. Now we begin to evaluate it.

```
3> (intloopProcess 1 T
   (nullstream #0=next1 ncloopParse
   (nullstream #0# lineoftoks
   (nullstream incZip1 incRenumberLine
   (nullstream incLude1 0 ("1") 0 ("strings") (1))
   (nullstream incigen1 0))))))
```

`intloopProcess` calls `StreamNull` which walks the delay applying the second value, which is a function, to the rest of the delay. Thus, all of the functions we packaged into the delay will be evaluated.

The result of each function call, e.g the result of calling `next1` will be a pair, which we call a ParsePair. The car of the ParsePair is `rplaca` into the delay and the cdr of the ParsePair is `rplacd` into the delay. So the delay is gradually reduced by each function call.

```
4> (StreamNull
   (nullstream #0=next1 ncloopParse
   (nullstream #0# lineoftoks
   (nullstream incZip1 incRenumberLine
   (nullstream incLude1 0 ("1") 0 ("strings") (1))
   (nullstream incigen1 0))))))
```

Here we see the `next1` function being called from the delay. It immediately calls `StreamNull` to process the rest of the delay.
9.2. PARSING THE INPUT

StreamNull(p555), now working on the inner portion of the delay, finds the function next1(p263) and calls it, which results in an immediate inner call to StreamNull(p555).

Descending even further, the StreamNull(p555) finds incZip1(p299), which finds the function incRenumberLine(p300) and two delays.

incZip1(p299) invokes StreamNull(p555) on the first delay, which invokes incLude1(p306) on the rest of the delay.

incLude1(p306) unpacks the argument list and invokes StreamNull(p555) on the second argument ("1") which is not the expected symbol nonnullstream so StreamNull(p555) immediately returns NIL.

Next, incLude1(p306) calls incClassify(p325) to which calls incCommand?(p326) which checks for a leading ". Since there isn’t one incClassify(p325) immediately returns a list of NIL, 0, and the empty string.
12> (incClassify "1")
13> (incCommand? "1")
<12 (incCommand? NIL)
<12 (incClassify (NIL 0 "))

12> (skipping? 1)
13> (keepPart? 1)
<13 (keepPart? T)
<12 (skipping? NIL)

12> (xlok 0 "1" 1 "strings")
13> (xlk1 0 "1" 1 "strings")
14> (incline1 0 "1" "1" -1 1 "strings")
15> (increate 0 "1" -1 1 "strings")
15> (increate (0 "1" -1 1 "strings") . 1) . "1")
14 (incline1 (((0 "1" -1 1 "strings") . 1) . "1")
<13 (xlk1 (((0 "1" -1 1 "strings") . 1) . "1")
(NIL |none|))
<12 (xlk (((0 "1" -1 1 "strings") . 1) . "1")
(NIL |none|))

12> (inlude 0 NIL 1 ("strings") (1))
13> (delay |inlude1| (0 NIL 1 ("strings") (1)))
13> (delay |
nonnullstream| |inlude1| 0 NIL 1 ("strings") (1))
<12 (|inlude| |
nonnullstream| |inlude1| 0 NIL 1 ("strings") (1))
<11 (|inlude1| |
nonnullstream| |inlude1| 0 NIL 1 ("strings") (1))
<10 (|streamnull| NIL)

10> (|streamnull| (nonnullstream |incigen1| 0))
11> (|incigen1| 0)
12> (|incigen| 1)
13> (|delay| |incigen1| (1))
13> (|delay| (nonnullstream |incigen1| (1)))
12> (|incigen| (nonnullstream |incigen1| (1)))
11> (|incigen1| (1 |nonnullstream| |incigen1| 1))
10> (|streamnull| NIL)
10> (|increnumberline| (O "1" "1" "1") (NIL |none|))
11> (|increnumberitem| (O "1" "1" "1") (NIL |none|))
12> (|incsetglobalnum| (O "1" "1" "1") (NIL |none|))
11> (|increnumberitem| (O "1" "1" "1") (NIL |none|))
11> (|inclevelname| (O "1" "1" "1") (NIL |none|))
9.2. PARSING THE INPUT

<11 (|incHandleMessage| 0)
<10 (|incRenumberLine|
  ((0 "1" 1 "strings") . 1) . "1")

10> (|incZip| |incRenumberLine|
  (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
  (|nonnullstream| |incIgen1| 1))
11> (|Delay| |incZip1|
  (|incRenumberLine|
   (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
   (|nonnullstream| |incIgen1| 1)))
<11 (|Delay|
  (|nonnullstream| |incZip1| |incRenumberLine|
   (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
   (|nonnullstream| |incIgen1| 1)))
10> (|incZip|
  (|nonnullstream| |incZip1| |incRenumberLine|
   (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
   (|nonnullstream| |incIgen1| 1)))
9> (|incZip1|
  (((0 "1" 1 "strings") . 1) . "1")
  |nonnullstream| |incZip1| |incRenumberLine|
  (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
  (|nonnullstream| |incIgen1| 1)))
8> (|lineoftoks|
  (((0 "1" 1 "strings") . 1) . "1")
  |nonnullstream| |incZip1| |incRenumberLine|
  (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
  (|nonnullstream| |incIgen1| 1)))
9> (|nextline|
  (((0 "1" 1 "strings") . 1) . "1")
  |nonnullstream| |incZip1| |incRenumberLine|
  (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
  (|nonnullstream| |incIgen1| 1)))
10> (|npNull|
  (((0 "1" 1 "strings") . 1) . "1")
  |nonnullstream| |incZip1| |incRenumberLine|
  (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
  (|nonnullstream| |incIgen1| 1)))
11> (|StreamNull|
  (((0 "1" 1 "strings") . 1) . "1")
  |nonnullstream| |incZip1| |incRenumberLine|
  (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
  (|nonnullstream| |incIgen1| 1)))
<11 (|StreamNull| NIL)
<10 (|npNull| NIL)
CHAPTER 9. STARTING AXIOM

10> (STRPOSL " " "1" 0 T)
<10 (STRPOSL 0)
<9 (\nextline| T)
9> (\scanIgnoreLine| "1" 0)
<9 (\scanIgnoreLine| 0)
9> (\incPrefix?| "command" 1 "1")
<9 (\incPrefix?| NIL)
9> (\scanToken|)
10> (\startsComment?|)
<10 (\startsComment?| NIL)
10> (\startsNegComment?|)
<10 (\startsNegComment?| NIL)
10> (\punctuation?| 49)
<10 (\punctuation?| NIL)
10> (\digit?| \#1)
<10 (\digit?| 1)
10> (\scanNumber|)
11> (\spleI| \digit?)
12> (\spleIi| \digit?| NIL)
13> (\digit?| \#1)
14> (\DIGITP \#1)
<14 (\DIGITP 1)
<13 (\digit?| 1)
12> (\spleIi| "1")
<11 (\spleI| "1")
11> (\lfinteger| "1")
<11 (\lfinteger| (\integer| "1"))
<10 (\scanNumber| (\integer| "1"))
10> (\lnExtraBlanks| (0 "1" 1 1 "strings"))
<10 (\lnExtraBlanks| 0)
10> (\constoken|
"1" (0 "1" 1 1 "strings") (\integer| "1") 0)
11> (\ncPutQ|
 (\integer| . "1") |posn| ((0 "1" 1 1 "strings") . 0))
12> (\ncAlist| (\integer| . "1"))
<12 (\ncAlist| NIL)
12 (\ncAlist| (\integer| . "1"))
<12 (\ncAlist| NIL)
12> (\ncTag| (\integer| . "1"))
<11 (\ncPutQ| ((0 "1" 1 1 "strings") . 0))
<10 (\constoken|
 (((\integer| (|posn| (0 "1" 1 1 "strings") . 0)). "1"))
10> (\dqUnit|
 (((\integer| (|posn| (0 "1" 1 1 "strings") . 0)). "1")))
<10 (\dqUnit|)
9.2. PARSING THE INPUT

(#0=(((#0 ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"). #0#))

<9 (|scanToken|
    (#0=(((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"). #0#))

9> (|dqAppend| NIL
    (#0=(((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"). #0#))

<9 (|dqAppend|
    (#0=(((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"). #0#))

<8 (|lineoftoks|
    ((#0=(
        ((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0)) . "1") . #0#
    )
    )

<8 (|lineoftoks|
    ((#0=(
        ((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0)) . "1") . #0#
    )
    )

8> (|next| |lineoftoks|
    (|nonnullstream| |incZip1| |incRenumberLine|
        (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
        (|nonnullstream| |incIgen1| 1))

9> (|delay| |next1|
    (|lineoftoks|
        (|nonnullstream| |incZip1| |incRenumberLine|
            (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
            (|nonnullstream| |incIgen1| 1))

<9 (|delay| |next1| |lineoftoks|
    (|nonnullstream| |incZip1| |incRenumberLine|
        (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
        (|nonnullstream| |incIgen1| 1))

<8 (|next|
( |nullstream| |next1| |lineoftoks|
    (|nonnullstream| |incZip1| |incRenumberLine|
        (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
        (|nonnullstream| |incIgen1| 1)))

8> (|incAppend|
    ((#0=((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0)) . "1") . #0#)
    )

9> (|delay| |incAppend1|
    (((#1=(

9.2. PARSING THE INPUT

```
(\nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
(\nonnullstream| |incIgen1| 1))))

<8 (\StreamNull| NIL)

8> (\incAppend| NIL
(\nonnullstream| |next1| |lineoftoks|
(\nonnullstream| |incZip1| |incRenumberLine|
(\nonullstream| |incLude1| 0 NIL 1 ("strings") (1))
(\nonullstream| |incIgen1| 1))))

9> (\Delay| \incAppend1|
NIL
(\nonullstream| |next1| |lineoftoks|
(\nonullstream| |incZip1| |incRenumberLine|
(\nonullstream| |incLude1| 0 NIL 1 ("strings") (1))
(\nonullstream| |incIgen1| 1))))

<9 (\Delay|
(\nonullstream| |incAppend1| NIL
(\nonullstream| |next1| |lineoftoks|
(\nonullstream| |incZip1| |incRenumberLine|
(\nonullstream| |incLude1| 0 NIL 1 ("strings") (1))
(\nonullstream| |incIgen1| 1))))

<8 (\incAppend| (\nonullstream| |incAppend1| NIL
(\nonullstream| |next1| |lineoftoks|
(\nonullstream| |incZip1| |incRenumberLine|
(\nonullstream| |incLude1| 0 NIL 1 ("strings") (1))
(\nonullstream| |incIgen1| 1))))

<7 (\incAppend1|
(((\#0=(((\integer| (\posn| \#1=(0 "1" 1 1 "strings") . 0))
  . "1") . \#0#) (((\#1# . 1) . "1")
  . \#2=(\nonullstream| |incZip1| |incRenumberLine|
  (\nonullstream| |incLude1| 0 NIL 1 ("strings") (1))
  (\nonullstream| |incIgen1| 1))))
|\nonullstream| |incAppend1| NIL
(\nonullstream| |next1| |lineoftoks| \#2#)))

<6 (\StreamNull| NIL)

6> (\incLoopParse|
(((\#0=(((\integer| (\posn| \#1=(0 "1" 1 1 "strings") . 0))
  . "1") . \#0#) (((\#1# . 1) . "1")
  . \#2=(\nonullstream| |incZip1| |incRenumberLine|
  (\nonullstream| |incLude1| 0 NIL 1 ("strings") (1))
  (\nonullstream| |incIgen1| 1))))
|\nonullstream| |incAppend1| NIL
(\nonullstream| |next1| |lineoftoks| \#2#)))

7> (\incLoopDQlines|
(\#0=(((\integer| (\posn| \#1=(0 "1" 1 1 "strings") . 0))
  . "1") . \#0#) (((\#1# . 1) . "1")
  . \#2=(\nonullstream| |incZip1| |incRenumberLine|
  (\nonullstream| |incLude1| 0 NIL 1 ("strings") (1))
  (\nonullstream| |incIgen1| 1))))
```

CHAPTER 9. STARTING AXIOM

(|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
(|nonnullstream| |incIgen1| 1))

8> (|StreamNull|
  ((((0 "1" 1 1 "strings") . 1) . "1")
   |nonnullstream| |incZip1| |incRenumberLine|
   (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
   (|nonnullstream| |incIgen1| 1))}

<8 (|StreamNull| NIL)

8> (|tokPosn|
  (((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1")
   |nonnullstream| |incAlist|
   ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1")
   |nonnullstream| |incAlist|
   ((|posn| (0 "1" 1 1 "strings") . 0))}

<8 (|tokPosn| ((0 "1" 1 1 "strings") . 0))

8> (|poglobalLinePosn| ((0 "1" 1 1 "strings") . 0))
9> (|poglobalLinePosn| ((0 "1" 1 1 "strings") . 1))
9> (|poglobalLinePosn| ((0 "1" 1 1 "strings") . 1))
9> (|poglobalLinePosn| ((0 "1" 1 1 "strings"))
9> (|lnGlobalNum| 1)
<9 (|lnGlobalNum| 1)

8> (|poglobalLinePosn| NIL)

8> (|poglobalLinePosn| 1)

8> (|poglobalLinePosn| 1)

8> (|poglobalLinePosn| (0 "1" 1 1 "strings") . 1))
9> (|poglobalLinePosn| (0 "1" 1 1 "strings") . 1))
9> (|poglobalLinePosn| (0 "1" 1 1 "strings"))
9> (|poglobalLinePosn| (0 "1" 1 1 "strings"))
9> (|lnGlobalNum| (0 "1" 1 1 "strings"))
<9 (|lnGlobalNum| 1)

8> (|poglobalLinePosn| 1)

8> (|poglobalLinePosn| (0 "1" 1 1 "strings") . 1))
9> (|poglobalLinePosn| (0 "1" 1 1 "strings") . 1))
9> (|poglobalLinePosn| (0 "1" 1 1 "strings"))
9> (|lnGlobalNum| 1)
<9 (|lnGlobalNum| 1)

8> (|poglobalLinePosn| 1)

8> (|poglobalLinePosn| 1)
8> (|streamChop| 1
  ((((0 "1" 1 1 "strings") . 1) . "1")
   |nonnullstream| |incZip1| |incRenumberLine|
   (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
   (|nonnullstream| |incIgen1| 1))
9> (|StreamNull|
  (((0 "1" 1 1 "strings") . 1) . "1")
   |nonnullstream| |incZip1| |incRenumberLine|
   (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
   (|nonnullstream| |incIgen1| 1))

<9 (|StreamNull| NIL)

9> (|streamChop| 0
  ((|nonnullstream| |incZip1| |incRenumberLine|
    (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
    (|nonnullstream| |incIgen1| 1)))
10> (|StreamNull|
    ((|nonnullstream| |incZip1| |incRenumberLine|
      (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
      (|nonnullstream| |incIgen1| 1)))
11> (|incZip1| |incRenumberLine|
    (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))
    (|nonnullstream| |incIgen1| 1))
12> (|StreamNull|
    (|nonnullstream| |incLude1| 0 NIL 1 ("strings") (1))))
9.2. PARSING THE INPUT

```
13> (|include1| 0 NIL 1 ("strings") (1))
14> (|StreamNull| NIL)
<14 (|StreamNull| T)
14> (|Top?| 1)
<14 (|Top?| T)
<13 (|include1| (nullstream))
<12 (|StreamNull| T)
<11 (|incZip1| (nullstream))
<10 (|StreamNull| T)
<9 (|streamChop| (NIL NIL))
9> (|ncloopPrefix?| ")command" "1")
<9 (|ncloopPrefix?| NIL)
<8 (|streamChop| (((((0 "1" 1 1 "strings") . 1) . "1") NIL))
<7 (|ncloopDQlines| (((((0 "1" 1 1 "strings") . 1) . "1") NIL))
7> (|dqToList|)
    (#0=((
        (|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1"). #0#))
<7 (|dqToList|
    (((|integer| (|posn| (0 "1" 1 1 "strings") . 0)). "1")))
7> (|npParse|
    (((|integer| (|posn| (0 "1" 1 1 "strings") . 0)). "1")))
8> (|npFirstTok|
    (|tokPart|
        (((|integer| (|posn| (0 "1" 1 1 "strings") . 0)). "1"))
    (|tokPart| "1")
8> (|npItem|
    9> (|npQualDef|
        10> (|npComma|
            11> (|npTuple| |npQualifiedDefinition|)
        12> (|npListofFun|
            |npQualifiedDefinition|
            |npCommaBackSet|
            |pfTupleListOf|)
        13> (|npQualifiedDefinition|)
    14> (|npQualified| |npDefinitionOrStatement|)
    15> (|npDefinitionOrStatement|)
    16> (|npBackTrack| |npGives| DEF |npDef|
        17> (|npState|
            <17 (|npState|
                (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
                    . "1")))
            17> (|npGives|
                18> (|npBackTrack| |npExit| GIVES |npLambda|)
        19> (|npState|
            <19 (|npState|
                (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
                    . "1"))))
    19> (|npExit|)
```
CHAPTER 9. STARTING AxiOM

20> (|npBackTrack| |npAssign| EXIT |npFileExit|)
21> (|npState|)
<21 (|npState|
    (((((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1"))))
21> (|npAssign|)
22> (|npBackTrack| |npMDEF| BECOMES |npAssignment|)
23> (|npState|)
<23 (|npState|
    (((((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1"))))
23> (|npMDEF|)
24> (|npBackTrack| |npStatement| MDEF |npMDEFinition|)
25> (|npState|)
<25 (|npState|
    (((((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1"))))
25> (|npStatement|)
26> (|npExpress|)
27> (|npExpress1|)
28> (|npConditionalStatement|)
29> (|npConditional| |npQualifiedDefinition|)
30> (|npEqKey| IF)
<30 (|npEqKey| NIL)
<29 (|npConditional| NIL)
<28 (|npConditionalStatement| NIL)
28> (|npADD|)
29> (|npType|)
30> (|npMatch|)
31> (|npLeftAssoc| (IS ISNT) |npSuch|)
32> (|npSuch|)
33> (|npLeftAssoc| (BAR) |npLogical|)
34> (|npLogical|)
35> (|npLeftAssoc| (OR) |npDisjand|)
36> (|npDisjand|)
37> (|npLeftAssoc| (AND) |npDiscrim|)
38> (|npDiscrim|)
39> (|npLeftAssoc| (CASE HAS) |npQuiver|)
40> (|npQuiver|)
41> (|npRightAssoc| (ARROW LARROW) |npRelation|)
42> (|npState|)
<42 (|npState|
    (((((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
        . "1"))))
42> (|npRelation|)
43> (|npLeftAssoc|
    (EQUAL NOTEQUAL LT LE GT GE OANGLE CANGLE)
    |npSynthetic|)
44> (|npSynthetic|)
45> (|npBy|)
46> (|inpLeftAssoc| (BY) |npInterval|)
47> (|npInterval|)
48> (|inpArith|)
49> (|inpLeftAssoc| (MOD) |npSum|)
50> (|npSum|)
51> (|inpLeftAssoc| (PLUS MINUS) |npTerm|)
52> (|npTerm|)
53> (|inpInfGeneric| (MINUS PLUS))
54> (|inpDDInfKey| (MINUS PLUS))
55> (|inpInfKey| NIL)
<55 (|inpInfKey| NIL)
55> (|inpState|)
<55 (|inpState| (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
 . "1")))
55> (|inpEqKey| |'|)
<55 (|inpEqKey| NIL)
55> (|inpRestore|
 ((((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
 . "1")))
56> (|inpFirstTok|)
57> (|tokPart|
 (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
 . "1"))
<57 (|tokPart| "1")
<56 (|inpFirstTok| "1")
<55 (|inpRestore| T)
55> (|inpEqKey| BACKQUOTE)
<55 (|inpEqKey| NIL)
55> (|inpRestore|
 ((((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
 . "1")))
56> (|inpFirstTok|)
57> (|tokPart|
 (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
 . "1"))
<57 (|tokPart| "1")
<56 (|inpFirstTok| "1")
<55 (|inpRestore| T)
<54 (|inpDDInfKey| NIL)
<53 (|inpInfGeneric| NIL)
53> (|inpRemainder|)
54> (|inpLeftAssoc| (REM QUO) |npProduct|)
55> (|npProduct|)
56> (|inpLeftAssoc| (TIMES SLASH BACKSLASH SLASHSLASH BACKSLASHBACKSLASH SLASHBACKSLASH BACKSLASHSLASH)
 |npPower|)
57> (|npPower|)
58> (|inpRightAssoc| (POWER CARAT) |npColon|)
CHAPTER 9. STARTING AXIOM

59> (inpState)
<59 (inpState)
  (((((integer) (posn) (0 "1" 1 "strings") . 0))
   . "1")))
59> (inpColon)
60> (inpTypified)
61> (inpApplication)
62> (inpDotted inpPrimary)
63> (inpPrimary)
64> (inpPrimary1)
65> (inpEncAp inpAtom1)
66> (inpAtom1)
67> (inpPDefinition)
68> (inpParenthesized inpDefinitionlist)
69> (inpParenthesize ((|)) inpDefinitionlist)
70> (inpEqKey |(())
<70 (inpEqKey NIL)
<69 (inpParenthesize NIL)
69> (inpParenthesize |(\|| |\|)\inpDefinitionlist)
70> (inpEqKey |(\||)
<70 (inpEqKey NIL)
<69 (inpParenthesize NIL)
68> (inpParenthesized NIL)
67> (inpPDefinition NIL)
67> (inpName)
68> (inpId)
<68 (inpId NIL)
68> (inpSymbolVariable)
69> (inpState)
<69 (inpState)
  (((((integer) (posn) (0 "1" 1 "strings") . 0))
   . "1")))
69> (inpEqKey BACKQUOTE)
<69 (inpEqKey NIL)
69> (inpRestore)
  (((((integer) (posn) (0 "1" 1 "strings") . 0))
   . "1")))
70> (inpFirstTok)
71> (tokPart)
  (((((integer) (posn) (0 "1" 1 "strings") . 0))
   . "1"))
<71 (tokPart "1")
<70 (inpFirstTok "1")
<69 (inpRestore T)
<68 (inpSymbolVariable NIL)
<67 (inpName NIL)
67> (inpConstTok)
68> (tokType)
  (((((integer) (posn) (0 "1" 1 "strings") . 0))
   . "1")))
9.2. PARSING THE INPUT

69> (incTag
   ((|integer| |posn| (0 "1" 1 1 "strings") . 0))
   . "1")

<69 (incTag| integer|
<68 (tokType| integer|
68> (inpPush
   ((|integer| |posn| (0 "1" 1 1 "strings") . 0))
   . "1")

<68 (inpPush
   (((|integer| |posn| (0 "1" 1 1 "strings") . 0))
   . "1")))

68> (inpNext[])
69> (inpFirstTok[])
70> (tokPosn
   ((|integer| |posn| (0 "1" 1 1 "strings") . 0))
   . "1")
71> (incList
   ((|integer| |posn| (0 "1" 1 1 "strings") . 0))
   . "1")

<71 (incList| (|posn| (0 "1" 1 1 "strings") . 0))
<70 (tokPosn| ((0 "1" 1 1 "strings") . 0))
70> (tokConstruct| ERROR NOMORE
   ((0 "1" 1 1 "strings") . 0))
71> (pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
72> (poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<72 (poNoPosition?| NIL)
<71 (pfNoPosition?| NIL)
71> (ncPutQ| (ERROR . NOMORE) |posn|
   ((0 "1" 1 1 "strings") . 0))
72> (incList| (ERROR . NOMORE))
<72 (incList| NIL)
72> (incList| (ERROR . NOMORE))
<72 (incList| NIL)
72> (incTag| (ERROR . NOMORE))
<72 (incTag| ERROR)
<71 (ncPutQ| ((0 "1" 1 1 "strings") . 0))
<70 (tokConstruct
   ((ERROR |posn| (0 "1" 1 1 "strings") . 0))
   . NOMORE))
70> (tokPart
   ((ERROR |posn| (0 "1" 1 1 "strings") . 0)). NOMORE))
<70 (tokPart| NOMORE)
<69 (inpFirstTok| NOMORE)
<68 (inpNext| NOMORE)
<67 (inpConstTok| NOMORE)
67> (inpFromdom[])
68> (inpEqKey| $)
<68 (inpEqKey| NIL)
<67 (inpFromdom| T)
<66 (inpAtom1| T)
CHAPTER 9. STARTING AXIOM

66> (npAnyNo| npEnc1) 
67> (npEnc1) 
68> (npBDefinition) 
69> (npPDefinition) 
70> (npParenthesized| npDefinitionlist) 
71> (npParenthesize| (||) npDefinitionlist) 
72> (npEqKey| () 
72> (npEqKey| NIL) 
71> (npParenthesize| NIL) 
72> (npParenthesize| (||) npDefinitionlist) 
72> (npEqKey| (||) 
72> (npEqKey| NIL) 
71> (npParenthesize| NIL) 
70> (npParenthesized| NIL) 
69> (npPDefinition| NIL) 
69> (npBracketed| npDefinitionlist) 
70> (npParened| npDefinitionlist) 
71> (npEnclosed| (||) npBracket| npDefinitionlist) 
72> (npEqKey| (||) 
72> (npEqKey| NIL) 
71> (npEnclosed| NIL) 
71> (npEnclosed| (||) npBracket| npDefinitionlist) 
72> (npEqKey| (||) 
72> (npEqKey| NIL) 
71> (npEnclosed| NIL) 
70> (npParened| NIL) 
70> (npBracked| npDefinitionlist) 
71> (npEnclosed| [ ] npBracket| npDefinitionlist) 
72> (npEqKey| () 
72> (npEqKey| NIL) 
71> (npEnclosed| NIL) 
71> (npEnclosed| [||] npBracketBar| npDefinitionlist) 
72> (npEqKey| [||] 
72> (npEqKey| NIL) 
71> (npEnclosed| NIL) 
70> (npBracked| NIL) 
70> (npBraced| npDefinitionlist) 
71> (npEnclosed| { } npBrace| npDefinitionlist) 
72> (npEqKey| { } 
72> (npEqKey| NIL) 
71> (npEnclosed| NIL) 
71> (npEnclosed| {||} npBraceBar| npDefinitionlist) 
72> (npEqKey| {||} 
72> (npEqKey| NIL) 
71> (npEnclosed| NIL) 
70> (npBraced| NIL) 
70> (npAngleBared| npDefinitionlist) 
71> (npEnclosed| |<|| ||>|| npHide| npDefinitionlist)
9.2. PARSING THE INPUT

```lisp
72> (|npEqKey| |<\]|)
72> (|npEqKey| NIL)
71> (|npEnclosed| NIL)
70> (|npAngleBared| NIL)
69> (|npBracketed| NIL)
68> (|npBDefinition| NIL)
67> (|npEnc1| NIL)
66> (|npAnyNo| T)
66> (|npFromdom|)
67> (|npEqKey| $)
67> (|npEqKey| NIL)
66> (|npFromdom| T)
65> (|npEncAp| T)
64> (|npPrimary1| T)
63> (|npAnyNo| |npSelector|)
64> (|npSelector|)
65> (|npEqKey| DOT)
64> (|npSelector| NIL)
63> (|npAnyNo| T)
62> (|npDotted| T)
62> (|npApplication2|)
63> (|npDotted| |npPrimary1|)
64> (|npPrimary1|)
65> (|npEncAp| |npAtom1|)
66> (|npAtom1|)
67> (|npPDefinition|)
68> (|npParenthesized| |npDefinitionlist|)
69> (|npParenthesize| |(| |)| |npDefinitionlist|)
70> (|npEqKey| |(|||)
69> (|npEqKey| NIL)
69> (|npParenthesize| NIL)
69> (|npParenthesize| |(|\| |\|)| |npDefinitionlist|)
70> (|npEqKey| |(|\||)
70> (|npEqKey| NIL)
69> (|npParenthesize| NIL)
68> (|npParenthesized| NIL)
67> (|npPDefinition| NIL)
67> (|nPname|)
68> (|npId|)
68> (|npId| NIL)
68> (|npSymbolVariable|)
69> (|npState|)
69> (|npState|
  \N (\N (\|integer| (\|posn| (0 "1" 1 1 "strings") . 0))
  . "1")))
69> (|npEqKey| BACKQUOTE)
69> (|npEqKey| NIL)
69> (|npRestore|
```
(NIL ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
 . "1")))
70> (|inpFirstTok|)
71> (|tokPosn|
  ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
72> (|ncAlist|
  ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<72 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
71> (|tokConstruct| ERROR NOMORE
  ((0 "1" 1 1 "strings") . 0))
72> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
73> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<73 (|poNoPosition?| NIL)
72> (|ncPutQ| (ERROR . NOMORE) |posn|
  ((0 "1" 1 1 "strings") . 0))
73> (|ncAlist| (ERROR . NOMORE))
73> (|ncAlist| NIL)
73> (|ncAlist| ERROR . NOMORE))
73> (|ncAlist| NIL)
73> (|ncTag| (ERROR . NOMORE))
73> (|ncTag| ERROR)
72> (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<71 (|tokConstruct|
  ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
71> (|tokPart|
  ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
71> (|tokPart| NOMORE)
70> (|inpFirstTok| NOMORE)
69> (|inpRestore| T)
68> (|inpSymbolVariable| NIL)
67> (|inpName| NIL)
67> (|inpConstTok|)
68> (|tokType|
  ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
69> (|ncTag|
  ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<69 (|ncTag| ERROR)
<68 (|tokType| ERROR)
68> (|inpEqPeek| ")")
68> (|inpEqPeek| NIL)
67> (|inpConstTok| NIL)
67> (|inpDollar|)
68> (|inpEqPeek| $)
68> (|inpEqPeek| NIL)
67> (|inpDollar| NIL)
67> (|inpBDefinition|)
68> (|inpPDefinition|)
69> (|inpParenthesized| |npDefinitionlist|)
9.2. PARSING THE INPUT

\[
\begin{align*}
70> &= (\text{npParenthesize} | ( | ) | \text{npDefinitionlist} |) \\
71> &= (\text{npEqKey} | ( | ) \\
<71 &= (\text{npEqKey} | \text{NIL}) \\
<70 &= (\text{npParenthesize} | \text{NIL}) \\
70> &= (\text{npParenthesize} | (\ | |) | \text{npDefinitionlist} |) \\
71> &= (\text{npEqKey} | (\ | ) \\
<71 &= (\text{npEqKey} | \text{NIL}) \\
<70 &= (\text{npParenthesize} | \text{NIL}) \\
<69 &= (\text{npParenthesized} | \text{NIL}) \\
<68 &= (\text{npPDefinition} | \text{NIL}) \\
68> &= (\text{npBracketed} | \text{npDefinitionlist} |) \\
69> &= (\text{npParened} | \text{npDefinitionlist} |) \\
70> &= (\text{npEnclosed} | [ | ] | \text{pfParen} | \text{npDefinitionlist} |) \\
71> &= (\text{npEqKey} | [ | ] \\
<71 &= (\text{npEqKey} | \text{NIL}) \\
<70 &= (\text{npEnclosed} | \text{NIL}) \\
<69 &= (\text{npParened} | \text{NIL}) \\
69> &= (\text{npBracked} | \text{npDefinitionlist} |) \\
70> &= (\text{npEnclosed} | [ | ] | \text{pfBracket} | \text{npDefinitionlist} |) \\
71> &= (\text{npEqKey} | [ ] \\
<71 &= (\text{npEqKey} | \text{NIL}) \\
<70 &= (\text{npEnclosed} | \text{NIL}) \\
70> &= (\text{npEnclosed} | [ | ] | \text{pfBracketBar} | \text{npDefinitionlist} |) \\
71> &= (\text{npEqKey} | [ | ] \\
<71 &= (\text{npEqKey} | \text{NIL}) \\
<70 &= (\text{npEnclosed} | \text{NIL}) \\
<69 &= (\text{npBracked} | \text{NIL}) \\
69> &= (\text{npBraced} | \text{npDefinitionlist} |) \\
70> &= (\text{npEnclosed} | \{ | \} | \text{pfBrace} | \text{npDefinitionlist} |) \\
71> &= (\text{npEqKey} | \{} \\
<71 &= (\text{npEqKey} | \text{NIL}) \\
<70 &= (\text{npEnclosed} | \text{NIL}) \\
70> &= (\text{npEnclosed} | \{ | | \} | \text{pfBraceBar} | \text{npDefinitionlist} |) \\
71> &= (\text{npEqKey} | \{} \\
<71 &= (\text{npEqKey} | \text{NIL}) \\
<70 &= (\text{npEnclosed} | \text{NIL}) \\
<69 &= (\text{npBraced} | \text{NIL}) \\
69> &= (\text{npAngleBared} | \text{npDefinitionlist} |) \\
70> &= (\text{npEnclosed} | < | > | \text{pfHide} | \text{npDefinitionlist} |) \\
71> &= (\text{npEqKey} | < | > \\
<71 &= (\text{npEqKey} | \text{NIL}) \\
<70 &= (\text{npEnclosed} | \text{NIL}) \\
<69 &= (\text{npAngleBared} | \text{NIL}) \\
<68 &= (\text{npBracketed} | \text{NIL})
\end{align*}
\]
CHAPTER 9. STARTING AXIOM

<67 (|inpDefinition| NIL)
<66 (|inpAtom1| NIL)
<65 (|inpEncAp| NIL)
65> (|inpLet|)
66> (|inpLetQualified| |inpDefinitionOrStatement|)
67> (|inpEqKey| LET)
<67 (|inpEqKey| NIL)
<66 (|inpLetQualified| NIL)
<65 (|inpLet| NIL)
65> (|inpFix|)
66> (|inpEqKey| FIX)
<66 (|inpEqKey| NIL)
<65 (|inpFix| NIL)
65> (|inpMacro|)
66> (|inpEqKey| MACRO)
<66 (|inpEqKey| NIL)
<65 (|inpMacro| NIL)
65> (|inpBPileDefinition|)
66> (|inpPileBracketed| |inpPileDefinitionlist|)
67> (|inpEqKey| SETTAB)
<67 (|inpEqKey| NIL)
<66 (|inpPileBracketed| NIL)
<65 (|inpBPileDefinition| NIL)
65> (|inpDefn|)
66> (|inpEqKey| DEFN)
<66 (|inpEqKey| NIL)
<65 (|inpDefn| NIL)
65> (|inpRule|)
66> (|inpEqKey| RULE)
<66 (|inpEqKey| NIL)
<65 (|inpRule| NIL)
<64 (|inpPrimary1| NIL)
<63 (|inpDotted| NIL)
<62 (|inpApplication2| NIL)
<61 (|inpApplication| T)
61> (|inpAnyNo| |inpTypeStyle|)
62> (|inpTypeStyle|)
63> (|inpCoerceTo|)
64> (|inpTypedForm| COERCE |pfCoerceto|)
65> (|inpEqKey| COERCE)
<65 (|inpEqKey| NIL)
<64 (|inpTypedForm| NIL)
<63 (|inpCoerceTo| NIL)
63> (|inpRestrict|)
64> (|inpTypedForm| AT |pfRestrict|)
65> (|inpEqKey| AT)
<65 (|inpEqKey| NIL)
<64 (|inpTypedForm| NIL)
<63 (|inpRestrict| NIL)
63> (|inpPretend|)
9.2. PARSING THE INPUT

64> (|inpTypedForm| PRETEND |pfPretend|)
65> (|inpEqKey| PRETEND)
<65 (|inpEqKey| NIL)
<64 (|inpTypedForm| NIL)
<63 (|inpPretend| NIL)
63> (|inpColonQuery|)
64> (|inpTypedForm| ATAT |pfRetractTo|)
65> (|inpEqKey| ATAT)
<65 (|inpEqKey| NIL)
<64 (|inpTypedForm| NIL)
<63 (|inpColonQuery| NIL)
<62 (|inpTypeStyle| NIL)
<61 (|inpAnyNo| T)
<60 (|inpTypified| T)
60> (|inpAnyNo| |npTagged|)
61> (|npTagged|)
62> (|inpTypedForm1| COLON |pfTagged|)
63> (|inpEqKey| COLON)
<63 (|inpEqKey| NIL)
<62 (|inpTypedForm1| NIL)
<61 (|npTagged| NIL)
<60 (|inpAnyNo| T)
<59 (|inpColon| T)
59> (|inpInfGeneric| |POWER CARAT|)
60> (|inpDDInfKey| |POWER CARAT|)
61> (|inpInfKey| |POWER CARAT|)
<61 (|inpInfKey| NIL)
61> (|inpState|)
<61 (|inpState|)

(NIL
  ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
   "1")))
61> (|inpEqKey| |'|)
<61 (|inpEqKey| NIL)
61> (|npRestore|)
(NIL
  ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
   "1")))
62> (|inpFirstTok|)
63> (|tokPosn|)
  ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE)
64> (|incAlist|
  ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE)
<64 (|incAlist| ((|posn| (0 "1" 1 1 "strings") . 0))
<63 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
63> (|tokConstruct| ERROR NOMORE
  ((0 "1" 1 1 "strings") . 0))
64> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
65> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<65 (|poNoPosition?| NIL)
<62 (inpFirstTok| NOMORE)
<61 (inpRestore| T)
<60 (inpDDInfKey| NIL)
<59 (inpInfGeneric| NIL)
<58 (inpRightAssoc| T)
<57 (inpPower| T)
57> (inpInfGeneric|
    (TIMES SLASH BACKSLASH SLASHSLASH BACKSLASHBACKSLASH SLASHBACKSLASH BACKSLASHSLASH))
58> (inpDDInfKey|
    (TIMES SLASH BACKSLASH SLASHSLASH BACKSLASHBACKSLASH SLASHBACKSLASH BACKSLASHSLASH))
59> (inpInfKey|
    (TIMES SLASH BACKSLASH SLASHSLASH BACKSLASHBACKSLASH SLASHBACKSLASH BACKSLASHSLASH))
<59 (inpInfKey| NIL)
59> (inpState)
<59 (inpState|
    (NIL
        (((integer| (posn| (0 "1" 1 1 "strings") . 0))
            . "1"))))
59> (inpEqKey| |')|
<59 (inpEqKey| NIL)
59> (inpRestore|
    (NIL
        (((integer| (posn| (0 "1" 1 1 "strings") . 0))
            . "1"))))
60> (inpFirstTok())
61> (|tokPosn|
    (((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
62> (|ncAlist|
    (((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<62 (|ncAlist| (((|posn| (0 "1" 1 1 "strings") . 0))))
61> (|tokConstruct| ERROR NOMORE
    (0 "1" 1 1 "strings") . 0))
62> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
63> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<63 (|poNoPosition?| NIL)
62> (|pfNoPosition?| NIL)
62> (|ncPutQ| (ERROR . NOMORE) |posn|
    (0 "1" 1 1 "strings") . 0))
63> (|ncAlist| (ERROR . NOMORE))
<63 (|ncAlist| NIL)
63> (|ncAlist| (ERROR . NOMORE))
<63 (|ncAlist| NIL)
63> (|ncTag| (ERROR . NOMORE))
<63 (|ncTag| ERROR)
<62 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<61 (|tokConstruct|
9.2. Parsing the Input

<57 (inpState
  (NIL
   (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
    . "1"))))
57> (inpEqKey| ')
<57 (inpEqKey| NIL)
57> (inpRestore)
  (NIL
   (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
    . "1")))
58> (inpFirstTok)
59> (tokPosn
  ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)). NOMORE))
60> (ncAlist
  ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)). NOMORE))
<60 (ncAlist
  ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)). NOMORE))
<59 (tokPosn
  (("1" 1 1 "strings") . 0))
59> (tokConstruct
   ERROR NOMORE
   (0 "1" 1 1 "strings") . 0))
60> (pfNoPosition?
   (("1" 1 1 "strings") . 0))
61> (poNoPosition?
   (("1" 1 1 "strings") . 0))
<61 (poNoPosition?
   NIL)
<60 (pfNoPosition?
   NIL)
60> (ncPutQ
   (ERROR . NOMORE)
   (posn
    ((0 "1" 1 1 "strings") . 0)))
61> (ncAlist
   (ERROR . NOMORE))
<61 (ncAlist
   (ERROR . NOMORE))
<61 (ncAlist
   NIL)
61> (ncTag
   (ERROR . NOMORE))
<61 (ncTag
   ERROR)
<60 (ncPutQ
   (("1" 1 1 "strings") . 0))
<59 (tokConstruct
   ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)). NOMORE))
59> (tokPart
   ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)). NOMORE))
<59 (tokPart
   NOMORE)
<58 (inpFirstTok
   NOMORE)
<57 (inpRestore
   T)
57> (inpEqKey
   BACKQUOTE)
<57 (inpEqKey
   NIL)
57> (inpRestore)
  (NIL
   (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
    . "1")))
58> (inpFirstTok)
59> (tokPosn
  ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)). NOMORE))
60> (ncAlist
  ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)). NOMORE))
CHAPTER 9. STARTING Axiom

<60 (incAlist (posn (0 "1" 1 1 "strings") . 0))
<59 (tokPosn ((0 "1" 1 1 "strings") . 0))
59> (tokConstruct ERROR NOMORE
   ((0 "1" 1 1 "strings") . 0))
60> (pfNoPosition? ((0 "1" 1 1 "strings") . 0))
61> (poNoPosition? ((0 "1" 1 1 "strings") . 0))
<61 (poNoPosition? NIL)
<60 (pfNoPosition? NIL)
60> (incPutQ (ERROR . NOMORE) posn
   ((0 "1" 1 1 "strings") . 0))
61> (incAlist (ERROR . NOMORE))
<61 (incAlist NIL)
61> (incAlist (ERROR . NOMORE))
<61 (incAlist NIL)
61> (incTag (ERROR . NOMORE))
61> (incTag ERROR)
60> (incPutQ ((0 "1" 1 1 "strings") . 0))
<59 (tokConstruct
   ((ERROR (posn (0 "1" 1 1 "strings") . 0)) . NOMORE))
59> (tokPart
   ((ERROR (posn (0 "1" 1 1 "strings") . 0)) . NOMORE))
<59 (tokPart NOMORE)
<58 (inpFirstTok NOMORE)
<57 (inpRestore T)
<56 (inpDDInfKey NIL)
<55 (inpInfGeneric NIL)
<54 (inpLeftAssoc T)
<53 (inpRemainder T)
<52 (inpTerm T)
52> (inpInfGeneric (PLUS MINUS))
53> (inpDDInfKey (PLUS MINUS))
54> (inpInfKey (PLUS MINUS))
<54 (inpInfKey NIL)
54> (inpState)
<54 (inpState NIL
   (NIL
    ((integer (posn (0 "1" 1 1 "strings") . 0))
     . "1"))))
54> (inpEqKey |)'
<54 (inpEqKey NIL)
54> (inpRestore
   (NIL
    ((integer (posn (0 "1" 1 1 "strings") . 0))
     . "1")))
55> (inpFirstTok)
56> (tokPosn
   ((ERROR (posn (0 "1" 1 1 "strings") . 0)) . NOMORE))
57> (incAlist
   ((ERROR (posn (0 "1" 1 1 "strings") . 0)) . NOMORE))
<57 (incAlist ((posn (0 "1" 1 1 "strings") . 0)))
9.2. PARSING THE INPUT

<56 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
56> (|tokConstruct| ERROR NOMORE
   ((0 "1" 1 1 "strings") . 0))
57> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
58> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<58 (|poNoPosition?| NIL)
<57 (|pfNoPosition?| NIL)
57> (|ncPutQ| (ERROR . NOMORE) |posn|
   ((0 "1" 1 1 "strings") . 0))
58> (|ncAlist| (ERROR . NOMORE))
<58 (|ncAlist| NIL)
58> (|ncAlist| (ERROR . NOMORE))
<58 (|ncAlist| NIL)
58> (|ncAlist| (ERROR . NOMORE))
<58 (|ncAlist| NIL)
58> (|ncAlist| (ERROR . NOMORE))
<58 (|ncAlist| NIL)
58> (|ncAlist| (ERROR . NOMORE))
<58 (|ncAlist| NIL)
58> (|ncTag| (ERROR . NOMORE))
<58 (|ncTag| ERROR)
<57 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
56> (|tokConstruct|
   ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
56> (|tokPart|
   ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
56> (|tokPart|
   (ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<56 (|tokPart| NOMORE)
<55 (|npFirstTok| NOMORE)
<54 (|npRestore| T)
54> (|npEqKey| BACKQUOTE)
<54 (|npEqKey| NIL)
54> (|npRestore|
   (NIL
    (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
      "1")
     )))
55> (|npFirstTok|)
56> (|tokPosn|
   ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
57> (|ncAlist|
   ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<57 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<56 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
56> (|tokConstruct| ERROR NOMORE
   ((0 "1" 1 1 "strings") . 0))
57> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
58> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<58 (|poNoPosition?| NIL)
<57 (|pfNoPosition?| NIL)
57> (|ncPutQ| (ERROR . NOMORE) |posn|
   ((0 "1" 1 1 "strings") . 0))
58> (|ncAlist| (ERROR . NOMORE))
<58 (|ncAlist| NIL)
58> (|ncAlist| (ERROR . NOMORE))
<58 (|ncAlist| NIL)
58> (|ncTag| (ERROR . NOMORE))
<58 (|ncTag| ERROR)
CHAPTER 9. STARTING Axiom

<57 (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
<56 (|tokConstruct|
    ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
 56> (|tokPart|
    ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<56 (|tokPart| NOMORE)
<55 (|inpFirstTok| NOMORE)
<54 (|inpRestore| T)
<53 (|inpDDInfKey| NIL)
<52 (|inpInfGeneric| NIL)
<51 (|inpLeftAssoc| T)
<50 (|inpSum| T)
 50> (|inpInfGeneric| (MOD))
 51> (|inpDDInfKey| (MOD))
 52> (|inpInfKey| NIL)
 52> (|inpState|)
 52> (|inpState|)
    (NIL
     (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1")))
 52> (|inpEqKey| |'|)
<52 (|inpEqKey| NIL)
<52 (|inpEqKey| NIL)
<51 (|inpRestore|)
    (NIL
     (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
       . "1")))
 53> (|inpFirstTok|)
 54> (|tokPosn|
    ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
 55> (|ncAlist|
    ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<55 (|ncAlist| (|posn| (0 "1" 1 1 "strings") . 0))
<54 (|tokPosn| (0 "1" 1 1 "strings") . 0))
 54> (|tokConstruct| ERROR NOMORE
    ((0 "1" 1 1 "strings") . 0))
 55> (|pfNoPosition?| (0 "1" 1 1 "strings") . 0))
 56> (|pNoPosition?| (0 "1" 1 1 "strings") . 0))
<56 (|pNoPosition?| NIL)
 55> (|pfNoPosition?| NIL)
 55> (|ncPutQ| (ERROR . NOMORE) |posn|
    ((0 "1" 1 1 "strings") . 0))
 56> (|ncAlist| (ERROR . NOMORE))
<56 (|ncAlist| NIL)
 56> (|ncAlist| (ERROR . NOMORE))
<56 (|ncAlist| NIL)
 56> (|ncTag| (ERROR . NOMORE))
<56 (|ncTag| ERROR)
<55 (|ncPutQ| (0 "1" 1 1 "strings") . 0))
<54 (|tokConstruct|
9.2. PARSING THE INPUT

((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))

54> (|tokPart|
   ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<54 (|tokPart| NOMORE)
<53 (|npFirstTok| NOMORE)
<52 (|inpRestore| T)
52> (|inpEqKey| BACKQUOTE)
<52 (|inpEqKey| NIL)
52> (|inpRestore|
   (NIL
    ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
     . "1"))))
53> (|inpFirstTok|)
54> (|tokPosn|
   ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
55> (|ncAlist|
   ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<55 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<54 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
54> (|tokConstruct| ERROR NOMORE
   ((0 "1" 1 1 "strings") . 0))
55> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
56> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<56 (|poNoPosition?| NIL)
<55 (|pfNoPosition?| NIL)
55> (|ncPutQ| (ERROR . NOMORE) |posn|
   ((0 "1" 1 1 "strings") . 0))
56> (|ncAlist| (ERROR . NOMORE))
<56 (|ncAlist| NIL)
56> (|ncAlist| (ERROR . NOMORE))
<56 (|ncAlist| NIL)
56> (|ncTag| (ERROR . NOMORE))
<56 (|ncTag| ERROR)
55> (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
54> (|tokConstruct|
   ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
54> (|tokPart|
   ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<54 (|tokPart| NOMORE)
<53 (|inpFirstTok| NOMORE)
<52 (|inpRestore| T)
<51 (|inpDDInfKey| NIL)
<50 (|inpInfGeneric| NIL)
<49 (|inpLeftAssoc| T)
<48 (|inpArith| T)
48> (|inpSegment|)
49> (|inpEqPeek| SEG)
<49 (|inpEqPeek| NIL)
<48 (|inpSegment| NIL)
<47 (|inpInterval| T)
9.2. PARSING THE INPUT

50> (inpFirstTok)
51> (tokPosn)
   (((ERROR (posn (0 "1" 1 1 "strings") . 0)) . NOMORE))
52> (ncAlist)
   (((ERROR (posn (0 "1" 1 1 "strings") . 0)) . NOMORE))
<52 (ncAlist) ((posn (0 "1" 1 1 "strings") . 0))
<51 (tokPosn) ((0 "1" 1 1 "strings") . 0))
51> (tokConstruct) ERROR NOMORE ((0 "1" 1 1 "strings") . 0))
52> (pfNoPosition?) ((0 "1" 1 1 "strings") . 0))
53> (poNoPosition?) ((0 "1" 1 1 "strings") . 0))
<53 (poNoPosition?) NIL)
<52 (pfNoPosition?) NIL)
52> (ncPutQ)
   (ERROR . NOMORE) |posn| ((0 "1" 1 1 "strings") . 0))
53> (ncAlist) (ERROR . NOMORE))
<53 (ncAlist) NIL)
53> (ncAlist) (ERROR . NOMORE))
<53 (ncAlist) NIL)
53> (ncTag) (ERROR . NOMORE))
<53 (ncTag) ERROR)
<52 (ncPutQ) ((0 "1" 1 1 "strings") . 0))
<51 (tokConstruct)
   (((ERROR (posn (0 "1" 1 1 "strings") . 0)) . NOMORE))
51> (tokPart)
   (((ERROR (posn (0 "1" 1 1 "strings") . 0)) . NOMORE))
<51 (tokPart) NOMORE)
<50 (inpFirstTok) NOMORE)
<49 (inpRestore) T)
<48 (inpDIDInfKey) NIL)
<47 (inpInfGeneric) NIL)
<46 (inpLeftAssoc) T)
<45 (inpBy) T)
45> (inpAmpersandFrom))
46> (inpAmpersand))
47> (inpEqKey) AMPERSAND)
<47 (inpEqKey) NIL)
<46 (inpAmpersand) NIL)
<45 (inpEqKey) AMPERSAND)
<44 (inpSynthetic) NIL)
44> (inpInfGeneric)
   (EQUAL NOTEQUAL LT LE GT GE OANGLE CANGLE))
45> (inpDIDInfKey)
   (EQUAL NOTEQUAL LT LE GT GE OANGLE CANGLE))
46> (inpInfKey) (EQUAL NOTEQUAL LT LE GT GE OANGLE CANGLE))
<46 (inpInfKey) NIL)
46> (inpState)
46> (inpState)
   NIL
   (((integer| (posn (0 "1" 1 1 "strings") . 0))
9.2. Parsing the Input

```
((0 "1" 1 1 "strings") . 0))
49> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
50> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<50 (|poNoPosition?| NIL)
<49 (|pfNoPosition?| NIL)
49> (|incPutQ| (ERROR . NOMORE) |posn|
 ((0 "1" 1 1 "strings") . 0))
50> (|incAlist| (ERROR . NOMORE))
<50 (|incAlist| NIL)
50> (|incAlist| (ERROR . NOMORE))
<50 (|incAlist| NIL)
50> (|incTag| (ERROR . NOMORE))
<50 (|incTag| ERROR)
<49 (|incPutQ| ((0 "1" 1 1 "strings") . 0))
<48 (|tokConstruct|
 ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
48> (|tokPart|
 ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<48 (|tokPart| NOMORE)
<47 (|inpFirstTok| NOMORE)
<46 (|inpRestore| T)
<45 (|inpDDInfKey| NIL)
<44 (|inpInfGeneric| NIL)
<43 (|inpLeftAssoc| T)
<42 (|inpRelation| T)
42> (|inpInfGeneric| (ARROW LARROW))
43> (|inpDDInfKey| (ARROW LARROW))
44> (|inpInfKey| (ARROW LARROW))
<44 (|inpInfKey| NIL)
44> (|inpState|)
44> (|inpState|
 (NIL
 ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
 . "1"))))
44> (|inpEqKey| |'|)
<44 (|inpEqKey| NIL)
44> (|inpRestore|
 (NIL
 ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
 . "1"))))
45> (|inpFirstTok|)
46> (|tokPosn|
 ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
47> (|incAlist|
 ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<47 (|incAlist| (|posn| (0 "1" 1 1 "strings") . 0))
<46 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
46> (|tokConstruct| ERROR NOMORE
 ((0 "1" 1 1 "strings") . 0))
47> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
```
CHAPTER 9. STARTING AXIOM

48> (poNoPosition? ((0 "1" 1 1 "strings") . 0))
<48 (poNoPosition? NIL)
<47 (pfNoPosition? NIL)
47> (ncPutQ (ERROR . NOMORE) |posn|
((0 "1" 1 1 "strings") . 0))
48> (ncAlist (ERROR . NOMORE))
<48 (ncAlist NIL)
48> (ncAlist (ERROR . NOMORE))
<48 (ncAlist NIL)
48> (ncAlist (ERROR . NOMORE))
<48 (ncAlist ERROR)
<47 (ncPutQ ((0 "1" 1 1 "strings") . 0))
<46 (tokConstruct
 ((ERROR (posn (0 "1" 1 1 "strings") . 0)). NOMORE))
46> (tokPart
 ((ERROR (posn (0 "1" 1 1 "strings") . 0)). NOMORE))
<46 (tokPart NOMORE)
<45 (inpFirstTok| NOMORE)
<44 (inpRestore | T)
44> (inpEqKey| BACKQUOTE)
<44 (inpEqKey NIL)
44> (inpRestore
 (NIL
 (((integer| (posn| (0 "1" 1 1 "strings") . 0))
 . "1"))))
45> (inpFirstTok|)
46> (tokPosn|
 ((ERROR (posn (0 "1" 1 1 "strings") . 0)). NOMORE))
47> (ncAlist |
 ((ERROR (posn (0 "1" 1 1 "strings") . 0)). NOMORE))
<47 (ncAlist |((posn| (0 "1" 1 1 "strings") . 0))
<46 (tokPosn| ((0 "1" 1 1 "strings") . 0))
46> (tokConstruct | ERROR NOMORE
 ((0 "1" 1 1 "strings") . 0))
47> (pfNoPosition? ((0 "1" 1 1 "strings") . 0))
48> (poNoPosition? ((0 "1" 1 1 "strings") . 0))
<48 (poNoPosition? NIL)
<47 (pfNoPosition? NIL)
47> (ncPutQ | ERROR . NOMORE |posn|
 ((0 "1" 1 1 "strings") . 0))
48> (ncAlist | ERROR . NOMORE))
<48 (ncAlist | NIL)
<48 (ncAlist | ERROR . NOMORE))
<48 (ncAlist | NIL)
48> (ncTag | ERROR)
<47 (ncPutQ | ((0 "1" 1 1 "strings") . 0))
<46 (tokConstruct
 ((ERROR (posn (0 "1" 1 1 "strings") . 0)). NOMORE))
46> (tokPart|
9.2. Parsing the Input

```lisp
((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<46 (|tokPart| NOMORE)
<45 (|inpFirstTok| NOMORE)
<44 (|inpRestore| T)
<43 (|inpDDInfKey| NIL)
<42 (|inpInfGeneric| NIL)
<41 (|inpRightAssoc| T)
<40 (|inpQuiver| T)
40> (|inpInfGeneric| (CASE HAS))
41> (|inpDDInfKey| (CASE HAS))
42> (|inpInfKey| (CASE HAS))
42> (|npState|)
42> (|npState| (NIL ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1")))
41> (|npEqKey| |'|)
42> (|npEqKey| NIL)
42> (|inpRestore| (NIL ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1")))
43> (|inpFirstTok|)
44> (|tokPosn|
  ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
45> (|incAlist|
  ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<45 (|incAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<44 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
44> (|tokConstruct| ERROR NOMORE
  ((0 "1" 1 1 "strings") . 0))
45> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
46> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<46 (|pfNoPosition?| NIL)
45> (|pfNoPosition?| NIL)
45> (|ncPutQ| (ERROR . NOMORE) |posn|
  ((0 "1" 1 1 "strings") . 0))
46> (|incAlist| (ERROR . NOMORE))
<46 (|incAlist| NIL)
46> (|incAlist| (ERROR . NOMORE))
<46 (|incAlist| NIL)
46> (|incTag| (ERROR . NOMORE))
<46 (|incTag| ERROR)
45> (|incPutQ| ((0 "1" 1 1 "strings") . 0))
44> (|tokConstruct|
  ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
44> (|tokPart|
  ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<44 (|tokPart| NOMORE)
```
CHAPTER 9. STARTING AXIOM

<43 (npFirstTok| NOMORE)
<42 (npRestore| T)
42> (npEqKey| BACKQUOTE)
<42 (npEqKey| NIL)
42> (npRestore|
  NIL
  (((integer| (posn| (0 "1" 1 1 "strings") . 0))
    . "1")))
43> (npFirstTok|
44> (tokPosn|
  (((ERROR (posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
45> (ncAlist|
  (((ERROR (posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<45 (ncAlist| ((posn| (0 "1" 1 1 "strings") . 0)))
44> (tokPosn| ((0 "1" 1 1 "strings") . 0))
44> (tokConstruct| ERROR NOMORE
  ((0 "1" 1 1 "strings") . 0))
45> (pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
46> (poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<46 (poNoPosition?| NIL)
45> (pfNoPosition?| NIL)
45> (ncPutQ| (ERROR . NOMORE) posn|
  ((0 "1" 1 1 "strings") . 0))
46> (ncAlist| (ERROR . NOMORE))
46> (ncAlist| NIL)
46> (ncAlist| (ERROR . NOMORE))
46> (ncAlist| NIL)
46> (ncPutQ| ((0 "1" 1 1 "strings") . 0))
45> (ncTag| (ERROR . NOMORE))
<44 (tokPart|
  (((ERROR (posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
44> (tokPart| NOMORE)
<44 (tokPart| NOMORE)
43> (npFirstTok| NOMORE)
42> (npRestore| T)
41> (npDDInfKey| NIL)
<40 (npInfGeneric| NIL)
<39 (npLeftAssoc| T)
<38 (npDiscrim| T)
38> (npInfGeneric| (AND))
39> (npDDInfKey| (AND))
40> (npInfKey| (AND))
<40 (npInfKey| NIL)
40> (npState)
<40 (npState|
  NIL
  (((integer| (posn| (0 "1" 1 1 "strings") . 0))
    . "1"))))
9.2. PARSING THE INPUT

```
40> (ifpEqKey ')
<40 (ifpEqKey NIL)
40> (ifpRestore)
   (NIL
     ((integer| (posn| (0 "1" 1 1 "strings") . 0))
      . "1")))
41> (ifpFirstTok)
42> (ifkPosn)
   ((ERROR (posn| (0 "1" 1 1 "strings") . 0)) . NOMORE)
43> (incAlist)
   ((ERROR (posn| (0 "1" 1 1 "strings") . 0)) . NOMORE)
<43 (incAlist| ((posn| (0 "1" 1 1 "strings") . 0))
<42 (ifkPosn| ((0 "1" 1 1 "strings") . 0))
42> (ifkConstruct| ERROR NOMORE ((0 "1" 1 1 "strings") . 0))
43> (ipfNoPosition?| ((0 "1" 1 1 "strings") . 0))
44> (ipoNoPosition?| ((0 "1" 1 1 "strings") . 0))
44> (ipfNoPosition?| NIL)
43> (ipfNoPosition?| NIL)
43> (incPutQ| (ERROR . NOMORE) |posn|
   ((0 "1" 1 1 "strings") . 0))
44> (incAlist| (ERROR . NOMORE))
<44 (incAlist| NIL)
44> (incAlist| (ERROR . NOMORE))
<44 (incAlist| NIL)
44> (incTag| (ERROR . NOMORE))
<44 (incTag| ERROR)
<43 (incPutQ| ((0 "1" 1 1 "strings") . 0))
42> (ifkPart)
   ((ERROR (posn| (0 "1" 1 1 "strings") . 0)) . NOMORE)
<42 (ifkPart| NOMORE)
41> (ifpFirstTok| NOMORE)
40> (ifpRestore| T)
40> (ifpEqKey| BACKQUOTE)
40> (ifpEqKey| NIL)
40> (ifpRestore)
   (NIL
     ((integer| (posn| (0 "1" 1 1 "strings") . 0))
      . "1"))
41> (ifpFirstTok)
42> (ifkPosn)
   ((ERROR (posn| (0 "1" 1 1 "strings") . 0)) . NOMORE)
43> (incAlist)
   ((ERROR (posn| (0 "1" 1 1 "strings") . 0)) . NOMORE)
<43 (incAlist| ((posn| (0 "1" 1 1 "strings") . 0))
<42 (ifkPosn| ((0 "1" 1 1 "strings") . 0))
42> (ifkConstruct| ERROR NOMORE
   ((0 "1" 1 1 "strings") . 0))
43> (ipfNoPosition?| ((0 "1" 1 1 "strings") . 0))
```
CHAPTER 9. STARTING AXIOM

44> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<44 (|poNoPosition?| NIL)
43> (|pfNoPosition?| NIL)
43> (|ncPutQ| (ERROR . NOMORE) |posn| ((0 "1" 1 1 "strings") . 0))
44> (|ncAlist| (ERROR . NOMORE))
44> (|ncAlist| (ERROR . NOMORE))
44> (|ncAlist| NIL)
44> (|ncTag| (ERROR . NOMORE))
44> (|ncTag| ERROR)
43> (|ncPutQ| ((0 "1" 1 1 "strings") . 0))
42> (|tokConstruct| ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
42> (|tokPart| ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
42> (|tokPart| NOMORE)
41> (|npFirstTok| NOMORE)
40> (|npRestore| T)
39> (|npDDInfKey| NIL)
38> (|npInfGeneric| NIL)
37> (|npLeftAssoc| T)
36> (|npDisjand| T)
35> (|npInfGeneric| (OR))
34> (|npDDInfKey| (OR))
33> (|npInfKey| (OR))
33> (|npState|)
32> (|npEqKey| |'|)
32> (|npRestore| (NIL ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1")))
31> (|npFirstTok|)
40> (|tokPosn| ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
41> (|ncAlist| ((ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<41 (|ncAlist| ((|posn| (0 "1" 1 1 "strings") . 0)))
<40 (|tokPosn| ((0 "1" 1 1 "strings") . 0))
40> (|tokConstruct| ERROR NOMORE ((0 "1" 1 1 "strings") . 0))
41> (|pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
42> (|poNoPosition?| ((0 "1" 1 1 "strings") . 0))
9.2. PARSING THE INPUT

<41 (\|pfNoPosition?\| NIL)
41> (\{ncPutQ\| (ERROR . NOMORE) \{posn\)
        ((0 "1" 1 1 "strings") . 0))
42> (\{ncAlist\| (ERROR . NOMORE))
<42 (\{ncAlist\| NIL)
42> (\{ncAlist\| (ERROR . NOMORE))
42> (\{ncAlist\| NIL)
42> (\{ncTag\| (ERROR . NOMORE))
42> (\{ncTag\| ERROR)
<41 (\{ncPutQ\| ((0 "1" 1 1 "strings") . 0))
<40 (\{tokConstruct|)
        ((ERROR (\{posn\| (0 "1" 1 1 "strings") . 0)). NOMORE))
40> (\{tokPart|)
        ((ERROR (\{posn\| (0 "1" 1 1 "strings") . 0)). NOMORE))
<40 (\{tokPart| NOMORE)
<39 (\{npFirstTok| NOMORE)
<38 (\{npRestore| T)
38> (\{npEqKey| BACKQUOTE)
38> (\{npEqKey| NIL)
38> (\{npRestore|)
        (NIL
          ((integer| (\{posn\| (0 "1" 1 1 "strings") . 0))
            . "1"))))
39> (\{npFirstTok|)
40> (\{tokPosn|)
        ((ERROR (\{posn\| (0 "1" 1 1 "strings") . 0)). NOMORE))
41> (\{ncAlist|)
        ((ERROR (\{posn\| (0 "1" 1 1 "strings") . 0)). NOMORE))
<41 (\{ncAlist\| (((\{posn\| (0 "1" 1 1 "strings") . 0))))
<40 (\{tokPosn| ((0 "1" 1 1 "strings") . 0))
40> (\{tokConstruct| ERROR NOMORE
        ((0 "1" 1 1 "strings") . 0))
41> (\|pfNoPosition?\| ((0 "1" 1 1 "strings") . 0))
42> (\|poNoPosition?\| ((0 "1" 1 1 "strings") . 0))
<42 (\|poNoPosition?\| NIL)
<41 (\|pfNoPosition?\| NIL)
41> (\{ncPutQ\| (ERROR . NOMORE) \{posn\)
        ((0 "1" 1 1 "strings") . 0))
42> (\{ncAlist\| (ERROR . NOMORE))
<42 (\{ncAlist\| NIL)
42> (\{ncAlist\| (ERROR . NOMORE))
42> (\{ncAlist\| NIL)
42> (\{ncTag\| (ERROR . NOMORE))
42> (\{ncTag\| ERROR)
<41 (\{ncPutQ\| ((0 "1" 1 1 "strings") . 0))
<40 (\{tokConstruct|)
        ((ERROR (\{posn\| (0 "1" 1 1 "strings") . 0)). NOMORE))
40> (\{tokPart|)
        ((ERROR (\{posn\| (0 "1" 1 1 "strings") . 0)). NOMORE))
<40 (\{tokPart| NOMORE)
CHAPTER 9. STARTING AXIOM

<39 (inpFirstTok| NOMORE)
<38 (inpRestore| T)
<37 (inpDDInfKey| NIL)
<36 (inpInfGeneric| NIL)
<35 (inpLeftAssoc| T)
<34 (inpLogical| T)
34> (inpInfGeneric| (BAR))
35> (inpDDInfKey| (BAR))
36> (inpInfKey| NIL)
36> (inpState|)
36> (inpState|
  (NIL
   (integer| (posn| (0 "1" 1 1 "strings") . 0))
    . "1"))
36> (inpEqKey| |'|)
<36 (inpEqKey| NIL)
36> (inpRestore|
  (NIL
   (integer| (posn| (0 "1" 1 1 "strings") . 0))
    . "1"))
37> (inpFirstTok|)
38> (tokPosn|
  ((ERROR (posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
39> (ncAlist|
  ((ERROR (posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<39 (ncAlist| ((posn| (0 "1" 1 1 "strings") . 0)))
<38 (tokPart| NOMORE)
<37 (npFirstTok| NOMORE)
<36 (inpRestore| T)
36> (inpEqKey| BACKQUOTE)
<36 (inpEqKey| NIL)
36> (inpRestore)
   (NIL
   (((integer| (posn| (0 "1" 1 1 "strings") . 0))
     . "1")))
37> (inpFirstTok)
38> (tokPosn|
   ((ERROR (posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
39> (incAlist|
   ((ERROR (posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<39 (incAlist| ((posn| (0 "1" 1 1 "strings") . 0)))
<38 (tokPosn| ((0 "1" 1 1 "strings") . 0))
38> (tokConstruct| ERROR NOMORE
   ((0 "1" 1 1 "strings") . 0))
39> (pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
40> (poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<40 (poNoPosition?| NIL)
<39 (pfNoPosition?| NIL)
39> (ncPutQ| (ERROR . NOMORE) |posn|
   ((0 "1" 1 1 "strings") . 0))
40> (ncAlist| (ERROR . NOMORE))
<40 (ncAlist| NIL)
40> (ncAlist| (ERROR . NOMORE))
<40 (ncAlist| NIL)
40> (ncTag| (ERROR . NOMORE))
<40 (ncTag| ERROR)
39> (ncPutQ| ((0 "1" 1 1 "strings") . 0))
<38 (tokConstruct|
   ((ERROR (posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
38> (tokPart|
   ((ERROR (posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<38 (tokPart| NOMORE)
37 (inpFirstTok| NOMORE)
36 (inpRestore| T)
<35 (inpDDInfKey| NIL)
<34 (inpInfGeneric| NIL)
<33 (inpLeftAssoc| T)
<32 (inpSuch| T)
32> (inpInfGeneric| (IS ISNT))
33> (inpDDInfKey| (IS ISNT))
34> (inpInfKey| (IS ISNT))
<34 (inpInfKey| NIL)
34> (inpState|)
<34 (inpState| NIL
   (NIL
   (((integer| (posn| (0 "1" 1 1 "strings") . 0))
     . "1")))
34> (inpEqKey| |'|)
<34 (inpEqKey| NIL)
CHAPTER 9. STARTING AXIOM

34> (npRestore)
   (NIL
    ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
     . "1")))
35> (npFirstTok)
36> (tokPosn)
   (ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE)
37> (incAlist)
   (ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE)
<37 (incAlist) ((|posn| (0 "1" 1 1 "strings") . 0))
<36 (tokPosn) ((0 "1" 1 1 "strings") . 0))
36> (tokConstruct| ERROR NOMORE
   ((0 "1" 1 1 "strings") . 0))
37> ( pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
38> ( poNoPosition?| ((0 "1" 1 1 "strings") . 0))
<38 (poNoPosition?| NIL)
<37 (pfNoPosition?| NIL)
37> (ncPutQ| (ERROR . NOMORE) |posn|
   ((0 "1" 1 1 "strings") . 0))
38> (ncAlist| (ERROR . NOMORE))
38> (ncAlist| NIL)
38> (ncAlist| ERROR . NOMORE))
38> (ncAlist| NIL)
38> (ncTag| (ERROR . NOMORE))
38> (ncTag| ERROR)
<37 (ncPutQ| ((0 "1" 1 1 "strings") . 0))
<36 (tokConstruct|
   (ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE)
36> (tokPart|
   (ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE)
<36 (tokPart| NOMORE)
35> (npFirstTok| NOMORE)
34> (npRestore| T)
34> (npEqKey| BACKQUOTE)
34> (npEqKey| NIL)
34> (npRestore|
   (NIL
    ((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
     . "1")))
35> (npFirstTok)
36> (tokPosn)
   (ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE)
37> (incAlist)
   (ERROR (|posn| (0 "1" 1 1 "strings") . 0)) . NOMORE)
<37 (incAlist) ((|posn| (0 "1" 1 1 "strings") . 0))
<36 (tokPosn) ((0 "1" 1 1 "strings") . 0))
36> (tokConstruct| ERROR NOMORE
   ((0 "1" 1 1 "strings") . 0))
37> ( pfNoPosition?| ((0 "1" 1 1 "strings") . 0))
38> ( poNoPosition?| ((0 "1" 1 1 "strings") . 0))
9.2. PARSING THE INPUT

<38 (\posNoPosition?| NIL)
<37 (\pfNoPosition?| NIL)
37> (\ncPutQ| (ERROR . NOMORE) \posn|
    ((0 "1" 1 1 "strings") . 0))
38> (\ncAlist| (ERROR . NOMORE))
<38 (\ncAlist| NIL)
38> (\ncAlist| (ERROR . NOMORE))
<38 (\ncAlist| NIL)
38> (\ncTag| (ERROR . NOMORE))
<38 (\ncTag| ERROR)
<37 (\ncPutQ| ((0 "1" 1 1 "strings") . 0))
<36 (\tokConstruct|
    ((ERROR (\posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
36> (\tokPart|
    ((ERROR (\posn| (0 "1" 1 1 "strings") . 0)) . NOMORE))
<36 (\tokPart| NOMORE)
<35 (\npFirstTok| NOMORE)
<34 (\npRestore| T)
<33 (\npDDInfKey| NIL)
<32 (\npInfGeneric| NIL)
<31 (\npLeftAssoc| T)
<30 (\npMatch| T)
30> (\npPop1|)
<30 (\npPop1|
    (((\integer| (\posn| (0 "1" 1 1 "strings") . 0))
    . "1")
30> (\npWith|
    (((\integer| (\posn| (0 "1" 1 1 "strings") . 0))
    . "1")
31> (\npEqKey| WITH)
<31 (\npEqKey| NIL)
<30 (\npWith| NIL)
30> (\npPush|
    (((\integer| (\posn| (0 "1" 1 1 "strings") . 0))
    . "1")
30> (\npPush|
    (((((\integer| (\posn| (0 "1" 1 1 "strings") . 0))
    . "1"))))
<29 (\npType|
    (((((\integer| (\posn| (0 "1" 1 1 "strings") . 0))
    . "1"))))
29> (\npPop1|)
<29 (\npPop1|
    (((\integer| (\posn| (0 "1" 1 1 "strings") . 0))
    . "1")
29> (\npAdd|
    (((\integer| (\posn| (0 "1" 1 1 "strings") . 0))
    . "1")
30> (\npEqKey| ADD)
<30 (\npEqKey| NIL)
CHAPTER 9. STARTING AXIOM

<29 (npAdd| NIL)
29> (npPush|
<br><29 (npAdd| NIL)
29> (npPush|
<br><29 (npPush|
<br><28 (npADD|
<br><27 (npExpress1|
<br><26 (npStatement| T)
<br><25 (npEqPeek| MDEF)
<br><24 (npBackTrack| T)
<br><23 (npMDEF| T)
<br><22 (npBackTrack| T)
<br><21 (npAssign| T)
<br><20 (npEqPeek| EXIT)
<br><19 (npEqPeek| NIL)
<br><18 (npBackTrack| T)
<br><17 (npBackTrack| T)
<br><16 (npBackTrack| T)
<br><15 (npDefinitionOrStatement| T)
<br><14 (npBackTrack| T)
9.2. PARSING THE INPUT

185

<13  (inpQualifiedDefinition| T)
13> (inpCommaBackSet|)
14> (inpEqKey| COMMA)
<14  (inpEqKey| NIL)
<13  (inpCommaBackSet| NIL)
<12  (inpListofFun| T)
<11  (inpTuple| T)
<10  (inpComma| T)
10>  (inpPop1|)
<10  (inpPop1|
  (((integer| (posn| (0 "1" 1 1 "strings") . 0))
   . "1")))
10>  (inpPush|
  (((((integer| (posn| (0 "1" 1 1 "strings") . 0))
   . "1"))))
<10  (inpPush|
  ((((((integer| (posn| (0 "1" 1 1 "strings") . 0))
   . "1")))))
<9  (inpQualDef|
  ((((integer| (posn| (0 "1" 1 1 "strings") . 0))
   . "1")))))
9>  (inpEqKey| SEMICOLON)
<9  (inpEqKey| NIL)
<9  (inpPop1|)
<9  (inpPop1|
  ((((integer| (posn| (0 "1" 1 1 "strings") . 0))
   . "1")))))
9>  (inpEnSequence|
  ((((integer| (posn| (0 "1" 1 1 "strings") . 0))
   . "1")))))
<9  (inpEnSequence|
  ((((integer| (posn| (0 "1" 1 1 "strings") . 0))
   . "1")))))
9>  (inpPush|
  (((((integer| (posn| (0 "1" 1 1 "strings") . 0))
   . "1")))
<9  (inpPush|
  ((((((integer| (posn| (0 "1" 1 1 "strings") . 0))
   . "1")))))
<8  (inpItem|
  (((((integer| (posn| (0 "1" 1 1 "strings") . 0))
   . "1")))))
<7  (inpParse|
  (((((integer| (posn| (0 "1" 1 1 "strings") . 0))
   . "1")))))
<6  (incloopParse|
  ((((((#0=0 "1" 1 1 "strings") . 1) . "1"))
  (((integer| (posn| #0# . 0)) . "1")))))
|nonnullstream| |incAppend1| NIL
|nonnullstream| |next1| |lineoftoks| (nullstream)))))
6>  (next| |ncloopParse|
  (nonnullstream| |incAppend1| NIL
  (nonnullstream| |next1| |lineoftoks| (nullstream))))
7>  (Delay| #0=|next1|
  (ncloopParse|
  (nonnullstream| |incAppend1| NIL
  (nonnullstream| |next1| |lineoftoks| (nullstream))))
(\begin{verbatim}
186

CHAPTER 9. STARTING AXIOM

(|nonnullstream| #0=|lineoftoks| (|nullstream|)))
<7 (|Delay|
  (|nonnullstream| #0=#|next1| |ncloopParse|
    (|nonnullstream| |incAppend1| NIL
     (|nonnullstream| #0=|lineoftoks| (|nullstream|))))

<6 (|next|
  (|nonnullstream| #0=#|next1| |ncloopParse|
    (|nonnullstream| |incAppend1| NIL
     (|nonnullstream| #0=|lineoftoks| (|nullstream|))))

6> (|incAppend|
  (((#0=(0 "1" 1 1 "strings") . 1) . "1")
    ((|integer| (|posn| #0# . 0)) . "1")))
  (|nonnullstream| #1=#|next1| |ncloopParse|
    (|nonnullstream| #1=|lineoftoks| (|nullstream|))))

7> (|Delay|
  #0=#|incAppend1|
  (((#2=(0 "1" 1 1 "strings") . 1) . "1")
    ((|integer| (|posn| #2# . 0)) . "1")))
  (|nonnullstream| #3=#|next1| |ncloopParse|
    (|nonnullstream| #3# |lineoftoks| (|nullstream|))))

<7 (|Delay|
  (|nonnullstream| #0=#|incAppend1|
    (((#2=(0 "1" 1 1 "strings") . 1) . "1")
     ((|integer| (|posn| #2# . 0)) . "1")))
    (|nonnullstream| #3=|next1| |ncloopParse|
     (|nonnullstream| #3# |lineoftoks| (|nullstream|))))

<6 (|incAppend|
  (|nonnullstream| #0=#|incAppend1|
    (((#2=(0 "1" 1 1 "strings") . 1) . "1")
     ((|integer| (|posn| #2# . 0)) . "1")))
    (|nonnullstream| #3=|next1| |ncloopParse|
     (|nonnullstream| #3# |lineoftoks| (|nullstream|))))

5> (|ncloopParse|
  (|nonnullstream| #0=(0 "1" 1 1 "strings") . 1) . "1")
  ((|integer| (|posn| #0# . 0)) . "1")
  (|nonnullstream| #1=|next1| |ncloopParse|
   (|nonnullstream| |incAppend1| NIL
    (|nonnullstream| #1# |lineoftoks| (|nullstream|))))
\end{verbatim}
9.2. PARSING THE INPUT

6> (|StreamNull| NIL)
6> (|incAppend| NIL
   (|nonnullstream| #0=|next1| |ncloopParse|
    (|nonnullstream| |incAppend1| NIL
     (|nonnullstream| #0# |lineoftoks| (|nullstream|))))))
7> (|Delay| #0=|incAppend1|
    NIL
    (|nonnullstream| #2=|next1| |ncloopParse|
     (|nonnullstream| #0# NIL
      (|nonnullstream| #2# |lineoftoks| (|nullstream|))))))
6> (|incAppend1|
    (|nonnullstream| #0=|incAppend1| NIL
     (|nonnullstream| #2=|next1| |ncloopParse|
      (|nonnullstream| #0# NIL
       (|nonnullstream| #2# |lineoftoks| (|nullstream|)))))))
5> (|incAppend1|
    (|nonnullstream| #1=|incAppend1| NIL
     (|nonnullstream| #2=|next1| |ncloopParse|
      (|nonnullstream| #0# NIL
       (|nonnullstream| #2# |lineoftoks| (|nullstream|))))))
4> (|pfAbSynOp?|
   (|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1") |command|
4> (|pfAbSynOp?| NIL)

4> (|intloopSpadProcess| 1
   (|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1")
   T)
5> (|ncPutQ| (|carrier|) |stepNumber| 1)
6> (|ncAlist| (|carrier|))
6> (|ncAlist| NIL)
6> (|ncAlist| (|carrier|))
6> (|ncTag| (|carrier|))
6> (|ncTag| |carrier|)
5> (|ncPutQ| (|carrier|) |stepNumber| 1)
6> (|ncAlist| (|carrier|) (|messages| NIL))
6> (|ncAlist| (|carrier|) (|messages| NIL))
6> (|ncAlist| (|carrier|) (|messages| NIL))

CHAPTER 9. STARTING AXIOM

6> (ncAlist| ((carrier| (stepNumber| . 1))))
<6 (ncAlist| ((stepNumber| . 1)))
6> (ncTag| ((carrier| (stepNumber| . 1))))
<6 (ncTag| |carrier|)
<5 (ncPutQ| NIL)
5> (ncPutQ|
 ((carrier| (messages|) (stepNumber| . 1)))
 |lines| (((#0=(0 "1" 1 1 "strings") . 1) . "1")))
6> (ncAlist| ((carrier| (messages|) (stepNumber| . 1))))
<6 (ncAlist| ((messages|) (stepNumber| . 1)))
6> (ncAlist| ((carrier| (messages|) (stepNumber| . 1))))
<6 (ncAlist| ((messages|) (stepNumber| . 1)))
6> (ncTag| ((carrier| (messages|) (stepNumber| . 1))))
<6 (ncTag| |carrier|)
<5 (ncPutQ| (((0 "1" 1 1 "strings") . 1) . "1")))
5> (intloopSpadProcess,interp|
 ((carrier| (lines| ((#0=(0 "1" 1 1 "strings") . 1) . "1"))
 (messages|) (stepNumber| . 1)))
((integer| (posn| #0# . 0)) . "1") T)
6> (ncConversationPhase| phParse|
 (((carrier| (lines| ((#0=(0 "1" 1 1 "strings") . 1) . "1"))
 (messages|) (stepNumber| . 1)))
((integer| (posn| #0# . 0)) . "1")))
7> (phParse|
 ((carrier| (lines| ((#0=(0 "1" 1 1 "strings") . 1) . "1"))
 (messages|) (stepNumber| . 1)))))
((integer| (posn| #0# . 0)) . "1"))
8> (ncPutQ|
 ((carrier| (lines| ((#0=(0 "1" 1 1 "strings") . 1) . "1")
 ) (messages|) (stepNumber| . 1)))
|ptree| ((integer| (posn| #0# . 0)) . "1"))
9> (ncAlist|
 (((carrier| (lines| ((#0=(0 "1" 1 1 "strings") . 1) . "1"))
 (messages|) (stepNumber| . 1)))))
<9 (ncAlist|
 (((lines| (((0 "1" 1 1 "strings") . 1) . "1"))
 (messages|) (stepNumber| . 1)))))
9> (ncAlist|
 (((carrier| (lines| (((0 "1" 1 1 "strings") . 1) . "1"))
 ) (messages|) (stepNumber| . 1))))
<9 (ncAlist|
 (((lines| (((0 "1" 1 1 "strings") . 1) . "1"))
 (messages|) (stepNumber| . 1)))
9> (ncTag|
 (((carrier| (lines| (((0 "1" 1 1 "strings") . 1) . "1"))
 ) (messages|) (stepNumber| . 1))))
<9 (ncTag| |carrier|)
<8 (ncPutQ|
 (((integer| (posn| (0 "1" 1 1 "strings") . 0)) . "1"))
<7 (phParse| OK)
9.2. PARSING THE INPUT

7> (|ncConversationPhase,wrapup|
   (||carrier|
      (|ptree|
         (|integer| (|posn| #0=(0 "1" 1 1 "strings") . 0))
         . "1")
         (|lines| ((#0# . 1) . "1")
         (|messages|
         (|stepNumber| . 1))))
<7 (|ncConversationPhase,wrapup| NIL)
<6 (|ncConversationPhase| OK)
6> (|ncConversationPhase| |phMacro|
   (||(carrier|
      (|ptree|
         (|integer| (|posn| #0=(0 "1" 1 1 "strings") . 0)) . "1")
         (|lines| ((#0# . 1) . "1")
         (|messages|
         (|stepNumber| . 1))))
7> (|phMacro|
   (||(carrier|
      (|ptree|
         (|integer| (|posn| #0=(0 "1" 1 1 "strings") . 0)) . "1")
         (|lines| ((#0# . 1) . "1")
         (|messages|
         (|stepNumber| . 1))))
8> (|ncEltQ|
   (||(carrier|
      (|ptree|
         (|integer| (|posn| #0=(0 "1" 1 1 "strings") . 0)) . "1")
         (|lines| ((#0# . 1) . "1")
         (|messages|
         (|stepNumber| . 1)))) |ptree|
9> (|ncAlist|
   (||(carrier|
      (|ptree|
         (|integer| (|posn| #0=(0 "1" 1 1 "strings") . 0))
         . "1")
         (|lines| ((#0# . 1) . "1")
         (|messages|
         (|stepNumber| . 1))))
<9 (|ncAlist|
   (||(carrier|
      (|ptree|
         (|integer| (|posn| #0=(0 "1" 1 1 "strings") . 0))
         . "1")
         (|lines| ((#0# . 1) . "1")
         (|messages|
         (|stepNumber| . 1))))
<8 (|ncEltQ|
   (||(integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1"))
8> (|ncPutQ|
   (||(carrier|
CHAPTER 9. STARTING AXIOM

({|ptree| .
  #0=(({|integer| ({|posn| #0=(0 "1" 1 1 "strings") . 0))
   . "1")
  ({|lines| ((#1# . 1) . "1")
  ({|messages|)
  ({|stepNumber| . 1}))
  |ptreePremacro| #0#)

 9> ({|ncAlist|
    (((|carrier|
    ({|ptree|
      ({|integer| ({|posn| #0=(0 "1" 1 1 "strings") . 0))
       . "1")
      ({|lines| ((#0# . 1) . "1")
      ({|messages|)
      ({|stepNumber| . 1})))))

 9> ({|ncAlist|
    (((|carrier|
    ({|ptree|
      ({|integer| ({|posn| #0=(0 "1" 1 1 "strings") . 0))
       . "1")
      ({|lines| ((#0# . 1) . "1")
      ({|messages|)
      ({|stepNumber| . 1})))))

 9> ({|ncAlist|
    (((|carrier|
    ({|ptree|
      ({|integer| ({|posn| #0=(0 "1" 1 1 "strings") . 0))
       . "1")
      ({|lines| ((#0# . 1) . "1")
      ({|messages|)
      ({|stepNumber| . 1})))))

 9> ({|ncTag|
    (((|carrier|
    ({|ptree|
      ({|integer| ({|posn| #0=(0 "1" 1 1 "strings") . 0))
       . "1")
      ({|lines| ((#0# . 1) . "1")
      ({|messages|)
      ({|stepNumber| . 1})))))

 9> ({|ncTag| |carrier|)

<8 ({|ncPutQ|
  ((|integer| ({|posn| (0 "1" 1 1 "strings") . 0)) . "1")

8> ({|macroExpanded|
  ((|integer| ({|posn| (0 "1" 1 1 "strings") . 0)) . "1")

9> ({|macExpand|}
9.2. PARSING THE INPUT

\[
\begin{align*}
&((\text{integer} ((\text{posn} (0 "1" 1 1 "strings" . 0)) . "1")) \\
&10> (\text{pfWhere}?) \quad ((\text{integer} ((\text{posn} (0 "1" 1 1 "strings" . 0)) . "1")) \\
&11> (\text{pfAbSynOp}?) \quad ((\text{integer} ((\text{posn} (0 "1" 1 1 "strings" . 0)) . "1")) \text{|Where|}) \\
&<11 (\text{pfAbSynOp}? \text{ NIL}) \\
&10 (\text{pfWhere}? \text{ NIL}) \\
&10> (\text{pfLambda}?) \quad ((\text{integer} ((\text{posn} (0 "1" 1 1 "strings" . 0)) . "1")) \\
&11> (\text{pfAbSynOp}?) \quad ((\text{integer} ((\text{posn} (0 "1" 1 1 "strings" . 0)) . "1")) \text{|Lambda|}) \\
&<11 (\text{pfAbSynOp}? \text{ NIL}) \\
&10 (\text{pfLambda}? \text{ NIL}) \\
&10> (\text{pfMacro}?) \quad ((\text{integer} ((\text{posn} (0 "1" 1 1 "strings" . 0)) . "1")) \\
&11> (\text{pfAbSynOp}?) \quad ((\text{integer} ((\text{posn} (0 "1" 1 1 "strings" . 0)) . "1")) \text{|Macro|}) \\
&<11 (\text{pfAbSynOp}? \text{ NIL}) \\
&10 (\text{pfMacro}? \text{ NIL}) \\
&10> (\text{pfId}?) \quad ((\text{integer} ((\text{posn} (0 "1" 1 1 "strings" . 0)) . "1")) \\
&11> (\text{pfAbSynOp}?) \quad ((\text{integer} ((\text{posn} (0 "1" 1 1 "strings" . 0)) . "1")) \text{|id|}) \\
&<11 (\text{pfAbSynOp}? \text{ NIL}) \\
&11> (\text{pfAbSynOp}?) \quad ((\text{integer} ((\text{posn} (0 "1" 1 1 "strings" . 0)) . "1")) \text{|iday|}) \\
&<11 (\text{pfAbSynOp}? \text{ NIL}) \\
&10 (\text{pfId}? \text{ NIL}) \\
&10> (\text{pfApplication}?) \quad ((\text{integer} ((\text{posn} (0 "1" 1 1 "strings" . 0)) . "1")) \\
&11> (\text{pfAbSynOp}?) \quad ((\text{integer} ((\text{posn} (0 "1" 1 1 "strings" . 0)) . "1")) \text{|Application|}) \\
&<11 (\text{pfAbSynOp}? \text{ NIL}) \\
&10 (\text{pfApplication}? \text{ NIL}) \\
&10> (\text{pfmapParts} [\text{macExpand}] \quad ((\text{integer} ((\text{posn} (0 "1" 1 1 "strings" . 0)) . "1")) \\
&11> (\text{pfLeaf}?) \quad ((\text{integer} ((\text{posn} (0 "1" 1 1 "strings" . 0)) . "1")))
\end{align*}
\]
9.2. PARSING THE INPUT

```lisp
(\langle \text{stepNumber} \rangle . 1)))
\langle 7 \langle \text{ncConversationPhase}, \text{wrapup} \rangle \text{ NIL}
\langle 6 \langle \text{ncConversationPhase} \rangle \text{ OK}
6\rangle \langle \text{ncConversationPhase} \rangle \langle \text{phIntReportMsgs} \rangle
\langle 5\rangle \langle \text{carrier} \rangle
\langle 4\rangle \langle \text{ptreePremacro} \rangle .
\langle 3\rangle \langle 0\rangle=((\langle \text{integer} \rangle \langle \text{posn} \rangle \langle 0 \rangle\text{ "1" 1 1 "strings"} . 0))
\langle 2\rangle \langle \text{integer} \rangle . "1"
\langle 1\rangle \langle \text{ptree} \rangle . #0#
\langle 0\rangle \langle \text{lines} \rangle \langle \#1\rangle . 1) . "1")
\langle 0\rangle \langle \text{messages} \rangle
\langle 0\rangle \langle \text{stepNumber} \rangle . 1))) \text{ T})
7\rangle \langle \text{phIntReportMsgs} \rangle
\langle 6\rangle \langle \text{carrier} \rangle
\langle 5\rangle \langle \text{ptreePremacro} \rangle .
\langle 4\rangle \langle 0\rangle=((\langle \text{integer} \rangle \langle \text{posn} \rangle \langle 0 \rangle\text{ "1" 1 1 "strings"} . 0))
\langle 3\rangle \langle \text{integer} \rangle . "1")
\langle 2\rangle \langle \text{ptree} \rangle . #0#
\langle 1\rangle \langle \text{lines} \rangle \langle \#1\rangle . 1) . "1")
\langle 0\rangle \langle \text{messages} \rangle
\langle 0\rangle \langle \text{stepNumber} \rangle . 1))) \text{ T})
8\rangle \langle \text{ncEltQ} \rangle
\langle 7\rangle \langle \text{carrier} \rangle
\langle 6\rangle \langle \text{ptreePremacro} \rangle .
\langle 5\rangle \langle 0\rangle=((\langle \text{integer} \rangle \langle \text{posn} \rangle \langle 0 \rangle\text{ "1" 1 1 "strings"} . 0))
\langle 4\rangle \langle \text{integer} \rangle . "1")
\langle 3\rangle \langle \text{ptree} \rangle . #0#
\langle 2\rangle \langle \text{lines} \rangle \langle \#1\rangle . 1) . "1")
\langle 1\rangle \langle \text{messages} \rangle
\langle 0\rangle \langle \text{stepNumber} \rangle . 1))) \text{ lines})
9\rangle \langle \text{ncAlist} \rangle
\langle 8\rangle \langle \text{carrier} \rangle
\langle 7\rangle \langle \text{ptreePremacro} \rangle .
\langle 6\rangle \langle 0\rangle=((\langle \text{integer} \rangle \langle \text{posn} \rangle \langle 0 \rangle\text{ "1" 1 1 "strings"} . 0))
\langle 5\rangle \langle \text{integer} \rangle . "1")
\langle 4\rangle \langle \text{ptree} \rangle . #0#
\langle 3\rangle \langle \text{lines} \rangle \langle \#1\rangle . 1) . "1")
\langle 2\rangle \langle \text{messages} \rangle
\langle 1\rangle \langle \text{stepNumber} \rangle . 1))) \langle \text{lines})
<9 \langle \text{ncAlist} \rangle
\langle 8\rangle \langle \text{carrier} \rangle
\langle 7\rangle \langle \text{ptreePremacro} \rangle .
\langle 6\rangle \langle 0\rangle=((\langle \text{integer} \rangle \langle \text{posn} \rangle \langle 0 \rangle\text{ "1" 1 1 "strings"} . 0))
\langle 5\rangle \langle \text{integer} \rangle . "1")
\langle 4\rangle \langle \text{ptree} \rangle . #0#
\langle 3\rangle \langle \text{lines} \rangle \langle \#1\rangle . 1) . "1")
\langle 2\rangle \langle \text{messages} \rangle
\langle 1\rangle \langle \text{stepNumber} \rangle . 1))) \langle \text{lines})
<8 \langle \text{ncEltQ} \rangle \langle 0 \rangle\text{ "1" 1 1 "strings"} . 1) . "1")
8\rangle \langle \text{ncEltQ} \rangle
\langle 7\rangle \langle \text{carrier} \rangle \langle \text{ptreePremacro} \rangle .
```
#0=((\integer\ (\posn\ #1=(0 "1" 1 1 "strings") . 0)) . "1")
(\ptree\ . #0#)
(\lines\ ((#1# . 1) . "1")
(\messages\)
(\stepNumber\ . 1)))

9> (\ncAlist|
(((\carrier|
  (\ptreePremacro\ .
    #0=((\integer\ (\posn\ #1=(0 "1" 1 1 "strings") . 0))
      . "1")
    (\ptree\ . #0#)
    (\lines\ ((#1# . 1) . "1")
    (\messages\)
    (\stepNumber\ . 1)))))

<9 (\ncAlist|
(((\ptreePremacro\ .
  #0=((\integer\ (\posn\ #1=(0 "1" 1 1 "strings") . 0))
    . "1")
  (\ptree\ . #0#)
  (\lines\ ((#1# . 1) . "1")
  (\messages\)
  (\stepNumber\ . 1))))

<8 (\ncEltQ\ NIL)
8> (\ncPutQ|
(((\carrier|
  (\ptreePremacro\ .
    #0=((\integer\ (\posn\ #1=(0 "1" 1 1 "strings") . 0))
      . "1")
    (\ptree\ . #0#)
    (\lines\ ((#1# . 1) . "1")
    (\messages\)
    (\stepNumber\ . 1)))) \ok?\ T)

9> (\ncAlist|
(((\carrier|
  (\ptreePremacro\ .
    #0=((\integer\ (\posn\ #1=(0 "1" 1 1 "strings") . 0))
      . "1")
    (\ptree\ . #0#)
    (\lines\ ((#1# . 1) . "1")
    (\messages\)
    (\stepNumber\ . 1))))

<9 (\ncAlist|
(((\ptreePremacro\ .
  #0=((\integer\ (\posn\ #1=(0 "1" 1 1 "strings") . 0))
    . "1")
  (\ptree\ . #0#)
  (\lines\ ((#1# . 1) . "1")
  (\messages\)})
(stepNumber . 1))

9> (ncAlist)
(((carrier)
  (ptreePremacro).
    #0=((integer) (#1=(0 "1" 1 1 "strings") . 0))
      . "1")
  (ptree . #0)
  (lines ((#1# . 1) . "1")
  (messages)
  (stepNumber . 1))))

<9 (ncAlist)
(((ptreePremacro).
    #0=((integer) (#1=(0 "1" 1 1 "strings") . 0))
      . "1")
  (ptree . #0)
  (lines ((#1# . 1) . "1")
  (messages)
  (stepNumber . 1))))

9> (ncTag)
(((carrier)
  (ptreePremacro).
    #0=((integer) (#1=(0 "1" 1 1 "strings") . 0))
      . "1")
  (ptree . #0)
  (lines ((#1# . 1) . "1")
  (messages)
  (stepNumber . 1))))

<9 (ncTag|carrier|)
<8 (ncPutQ| T)
<7 (phIntReportMsgs| OK)
7> (ncConversationPhase,wrapup)
(((carrier)
  (ok?| . T)
  (ptreePremacro).
    #0=((integer) (#1=(0 "1" 1 1 "strings") . 0))
      . "1")
  (ptree . #0)
  (lines ((#1# . 1) . "1")
  (messages)
  (stepNumber . 1))))

<7 (ncConversationPhase,wrapup| NIL)
<6 (ncConversationPhase| OK)
6> (ncConversationPhase| phInterpret)
(((carrier)
  (ok?| . T)
  (ptreePremacro).
    #0=((integer) (#1=(0 "1" 1 1 "strings") . 0))
      . "1")
  (ptree . #0)
  (lines ((#1# . 1) . "1")))
(|messages|)
(|stepNumber| . 1))))

8> (|phInterpret|
  (((|carrier|
    (|ok?| . T)
    (|ptreePremacro| .
      #0=((|integer| (|posn| #1=(0 "1 1 1 "strings") . 0))
       . "1"))
    (|ptree| . #0#
    (|lines| ((|#1# . 1) . "1"))
    (|messages|)
    (|stepNumber| . 1))))

8> (|ncEltQ|
  (((|carrier|
    (|ok?| . T)
    (|ptreePremacro| .
      #0=((|integer| (|posn| #1=(0 "1 1 1 "strings") . 0))
       . "1"))
    (|ptree| . #0#
    (|lines| ((|#1# . 1) . "1"))
    (|messages|)
    (|stepNumber| . 1))))

9> (|ncAlist|
  (((|carrier|
    (|ok?| . T)
    (|ptreePremacro| .
      #0=((|integer| (|posn| #1=(0 "1 1 1 "strings") . 0))
       . "1"))
    (|ptree| . #0#
    (|lines| ((|#1# . 1) . "1"))
    (|messages|)
    (|stepNumber| . 1))))

<9 (|ncEltQ|
  (((|carrier|
    (|ok?| . T)
    (|ptreePremacro| .
      #0=((|integer| (|posn| #1=(0 "1 1 1 "strings") . 0))
       . "1"))
    (|ptree| . #0#
    (|lines| ((|#1# . 1) . "1"))
    (|messages|)
    (|stepNumber| . 1))))

<9 (|incAlist|
  (((|carrier|
    (|ok?| . T)
    (|ptreePremacro| .
      #0=((|integer| (|posn| #1=(0 "1 1 1 "strings") . 0))
       . "1"))
    (|ptree| . #0#
    (|lines| ((|#1# . 1) . "1"))
    (|messages|)
    (|stepNumber| . 1))))

8> (|intInterpretPform|
  (((|integer| (|posn| (0 "1 1 1 "strings") . 0)) . "1")))

9> (|pf2Sex|
  (((|integer| (|posn| (0 "1 1 1 "strings") . 0)) . "1")))

10> (|pf2Sex1|
  (((|integer| (|posn| (0 "1 1 1 "strings") . 0) . "1")))
9.2. PARSING THE INPUT

. "1")
11> (|pfNothing?|
   (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
    . "1"))
12> (|pfAbSynOp?|
   (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
    . "1") |nothing|)
<12 (|pfAbSynOp?| NIL)
<11 (|pfNothing?| NIL)
11> (|pfSymbol?|
   (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
    . "1"))
12> (|pfAbSynOp?|
   (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
    . "1") |symbol|)
<12 (|pfAbSynOp?| NIL)
<11 (|pfSymbol?| NIL)
11> (|pfLiteral?|
   (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
    . "1"))
12> (|pfAbSynOp|
   (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
    . "1"))
<12 (|pfAbSynOp| |integer|)
<11 (|pfLiteral?|
   (((|integer| (|symbol| |expression| |one| |zero|
      |char| |string| |float|))
   . "1") |symbol| |expression| |one| |zero| |char| |string| |float|)
11> (|pfLiteral2Sex|
   (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
    . "1"))
12> (|pfLiteralClass|
   (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
    . "1"))
13> (|pfAbSynOp|
   (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
    . "1"))
<13 (|pfAbSynOp| |integer|)
<12 (|pfLiteralClass| |integer|)
12> (|pfLiteralString|
   (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
    . "1"))
13> (|tokPart|
   (((|integer| (|posn| (0 "1" 1 1 "strings") . 0))
    . "1"))
<13 (|tokPart| "1")
<12 (|pfLiteralString| "1")
<11 (|pfLiteral2Sex| 1)
<10 (|pf2Sex1| 1)
<9 (|pf2Sex| 1)
9> (|zeroOneTran| 1)
<9 (|zeroOneTran| 1)
9> (|processInteractive| 1
   ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1")
10> (PUT |algebra| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |algebra| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |analysis| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |analysis| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |coercion| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |coercion| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |compilation| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |compilation| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |debug| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |debug| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |evaluation| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |evaluation| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |gc| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |gc| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |history| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |history| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |instantiation| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |instantiation| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |load| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |load| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |modemaps| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |modemaps| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |optimization| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |optimization| |SpaceTotal| 0)
9.2. PARSING THE INPUT

9.2.1. Parsing the Input

<10 (PUT 0)
10> (PUT |querycoerce| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |querycoerce| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |other| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |other| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |diskread| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |diskread| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |print| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |print| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |resolve| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |resolve| |SpaceTotal| 0)
<10 (PUT 0)
10> (PUT |interpreter| |ClassTimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |interpreter| |ClassSpaceTotal| 0)
<10 (PUT 0)
10> (PUT |evaluation| |ClassTimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |evaluation| |ClassSpaceTotal| 0)
<10 (PUT 0)
10> (PUT |other| |ClassTimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |other| |ClassSpaceTotal| 0)
<10 (PUT 0)
10> (PUT |reclaim| |ClassTimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |reclaim| |ClassSpaceTotal| 0)
<10 (PUT 0)
10> (GETL |gc| |TimeTotal|)
<10 (GETL 0.0)
10> (PUT |gc| |TimeTotal| 0.050000000000000003)
<10 (PUT 0.050000000000000003)
10> (PUT |gc| |TimeTotal| 0.0)
<10 (PUT 0.0)
10> (PUT |gc| |SpaceTotal| 0)
<10 (PUT 0)
10> (|processInteractive1| 1
   ((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1")
11> (recordFrame |system|)
12> (|diffAlist| NIL NIL)
<12 (|diffAlist| NIL)
<11 (recordFrame NIL)
11> (GETL |other| |TimeTotal|)
11> (GETL 0.0)
11> (GETL |gc| |TimeTotal|)
11> (GETL 0.0)
11> (PUT |gc| |TimeTotal| 0.0)
11> (PUT 0.0)
11> (PUT |other| |TimeTotal| 0.0)
11> (PUT 0.0)
11> (interpretTopLevel 1
12> (((integer| ((posn (0 "1" 1 1 "strings") . 0)) . "1"))
13> (((integer| ((posn (0 "1" 1 1 "strings") . 0)) . "1"))
14> (((integer| ((posn (0 "1" 1 1 "strings") . 0)) . "1"))
14> (member| 1 (|noBranch| |noMapVal|))
14> (member| NIL)
14> (member| 1 (|nil| |true| |false|))
14> (member| NIL)
14> (member| |--immediateData--| NIL)
14> (member| NIL)
14> (isDomainValuedVariable| |--immediateData--|)
14> (isDomainValuedVariable| NIL)
14> (GETDATABASE |--immediateData--| CONSTRUCTOR)
14> (GETDATABASE NIL)
14> (GETDATABASE |--immediateData--| ABBREVIATION)
14> (GETDATABASE NIL)
14> (member| |--immediateData--|)
15> (getProplist| |--immediateData--| ((NIL)))
16> (search| |--immediateData--| (NIL))
16> (searchCurrentEnv| |--immediateData--| (NIL))
16> (searchTailEnv| |--immediateData--| NIL)
16> (searchTailEnv| NIL)
15> (search| NIL)
15> (search| |--immediateData--|)
16> (%|Category|
17> (modemap| (%|Category|) (%|Category|) (T *))
17> (%|join|)
17> (modemap|)
17> (%|Category|)
17> (%|Category|)
17> (%|Category|)
17> (%|Category|)
17> (T *)
17> (%|Category|)
17> (%|Category|)
17> (%|List| (|Category|))
17> (%|Category|)
9.2. Parsing the Input

```
16> (|searchCurrentEnv| |--immediateData--| NIL)
16> (|searchTailEnv| |--immediateData--| NIL)
<15 (|search| NIL)
<14 (|getProplist| NIL)
14> (|member| |--immediateData--| NIL)
14> (|member| NIL)
14> (|member| |--immediateData--| NIL)
14> (|member| NIL)
14> (|member| |--immediateData--| NIL)
14> (|member| NIL)
14> (|member| |--immediateData--| NIL)
14> (|member| NIL)
14> (|member| |--immediateData--| NIL)
14> (|member| NIL)
14> (|member| |--immediateData--| NIL)
14> (|member| NIL)
14> (|member| |--immediateData--| NIL)
14> (|member| NIL)
14> (|interpret2|)
   (#0=(|PositiveInteger|) . 1) #0#
   (((|integer| (|posn| (0 "1" 1 1 "strings") . 0)) . "1")
   1)
<14 (|interpret2| (((|PositiveInteger|) . 1))
<13 (|interpret1| (((|PositiveInteger|) . 1))
<12 (|interpret| (((|PositiveInteger|) . 1))
<11 (|interpretTopLevel| (((|PositiveInteger|) . 1))
11> (GETL |analysis| |TimeTotal|)
11> (GETL 0.0)
11> (GETL |gc| |TimeTotal|)
11> (GETL 0.0)
11> (PUT |gc| |TimeTotal| 0.0)
11> (PUT 0.0)
11> (PUT |analysis| |TimeTotal| 0.0)
11> (PUT 0.0)
11> (GETL |other| |TimeTotal|)
11> (GETL 0.0)
```
CHAPTER 9. STARTING AXIOM

11> (GETL |gc| |TimeTotal|)
<11 (GETL 0.0)
11> (PUT |gc| |TimeTotal| 0.0)
<11 (PUT 0.0)
11> (PUT |other| |TimeTotal| 0.0)
<11 (PUT 0.0)
11> (|recordAndPrint| 1 (|PositiveInteger|))

12> (GETDATABASE |PositiveInteger| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (|member| (|PositiveInteger|)
  ((|Mode|) (|Domain|) (|SubDomain| (|Domain|))))
<12 (|member| NIL)
12> (|member| (|PositiveInteger|)
  ((|Category|) (|Mode|) (|Domain|)
   (|SubDomain| (|Domain|))))
<12 (|member| NIL)
12> (GETL |print| |TimeTotal|)
<12 (GETL 0.0)
12> (GETL |gc| |TimeTotal|)
<12 (GETL 0.0)
12> (PUT |gc| |TimeTotal| 0.0)
<12 (PUT 0.0)
12> (PUT |print| |TimeTotal| 0.0)
<12 (PUT 0.0)
12> (|isEqualOrSubDomain| (|PositiveInteger|)
   (|OutputForm|))
<12 (|isEqualOrSubDomain| NIL)
12> (GETDATABASE |OutputForm| ABBREVIATION)
<12 (GETDATABASE OUTFORM)
12> (HPUT #<hash-table 00000000001ab1e40>
   (|OutputForm|) (1))
<12 (HPUT (1))
12> (HPUT #<hash-table 00000000001ab1e00>
   (NIL NIL NIL) (1 . T))
<12 (HPUT (1 . T))
12> (HPUT #<hash-table 00000000001ab1e00>
   (#0=(|OutputForm|) NIL NIL) (1 . #0#))
<12 (HPUT (1 |OutputForm|))
12> (HPUT #<hash-table 00000000001ab1e40>
   (|PositiveInteger|) (1))
<12 (HPUT (1))
12> (|member| (|OutputForm|) (((|Integer|) (|OutputForm|))))
<12 (|member| (((|OutputForm|))))
12> (|member| (|OutputForm|)
   (((|Mode|) (|Domain|) (|SubDomain| (|Domain|))))
<12 (|member| NIL)
12> (GETDATABASE |OutputForm| ABBREVIATION)
<12 (GETDATABASE OUTFORM)
12> (GETDATABASE |OutputForm| CDSIG)
9.2. PARSING THE INPUT

<12 (GETDATABASE (NIL))
12> (HPUT #<hash-table 0000000001ab1840>
   (|OutputForm|) (1 . T))
12> (HPUT (1 . T))
12> (|isPartialMode| (|OutputForm|))
12> (|isPartialMode| NIL)
12> (|member| |coerce| (= + * -))
12> (|member| NIL)
12> (|isPartialMode| (|OutputForm|))
12> (|isPartialMode| NIL)
12> (|member| |PositiveInteger| (|List| |Vector| |Stream| |FiniteSet| |Array|))
12> (|member| NIL)
12> (|member| |PositiveInteger| (|Union| |Record| |Mapping| |Enumeration|))
12> (|member| NIL)
12> (GETDATABASE |PositiveInteger| OPERATIONALIST)
12> (GETDATABASE
   (\(\leq\) ((|Boolean|) $ $) NIL))
   (|sample| ((($ NIL T CONST))
   (|recip| (((|Union| $ "failed") $) NIL))
   (|one?| (((|Boolean|) $) NIL))
   (|min| ((($ $ $) NIL))
   (|max| ((($ $ $) NIL))
   (|latex| (((|String|) $) NIL))
   (|hash| (((|SingleInteger|) $) NIL))
   (|gcd| ((($ $ NIL))
   (|coerce| (((|OutputForm|) $) NIL))
   (\(\ast\) ((($ $ (|NonNegativeInteger|)) NIL))
   ((($ (|PositiveInteger|)) NIL))
   (|One| ((($ NIL T CONST))
   (\(\geq\) (((|Boolean|) $ $) NIL))
   (> (((|Boolean|) $ $) NIL))
   (\(\leq\) (((|Boolean|) $ $) NIL))
   (\(\lt\) (((|Boolean|) $ $) NIL))
   (\(\lt\) (((|Boolean|) $ $) NIL))
   (+ ((($ $ NIL))
   (** ((($ (|NonNegativeInteger|)) NIL))
   ((($ (|PositiveInteger|)) NIL))
   (\(\ast\) (((|PositiveInteger|) $) NIL) ((($ $ NIL))))))
12> (|constructSubst| (|PositiveInteger|))
12> (|constructSubst| (($ |PositiveInteger|))
12> (|isEqualOrSubDomain| #0=(|PositiveInteger|) #0#)
12> (|isEqualOrSubDomain| T)
12> (|isEqualOrSubDomain| (|OutputForm|) (|OutputForm|))
12> (|isEqualOrSubDomain| T)
12> (|member| |OutputForm| (|Union| |Record| |Mapping| |Enumeration|))
12> (|member| NIL)
12> (GETDATABASE |OutputForm| OPERATIONALIST)
12> (GETDATABASE
9.2. PARSE THE INPUT

```
12> (constructSubst (OutputForm))
<12 (constructSubst ((OutputForm))))
12> (isEqualOrSubDomain (PositiveInteger) (OutputForm))
<12 (isEqualOrSubDomain NIL)
12> (HPUT #<hash-table 0000000001ab1e70> (coerce (OutputForm) (#0=([PositiveInteger])
(0#) NIL) (1 ([0# #1=(OutputForm) #0#] (#1# $) (NIL)))))
<12 (HPUT (1 (#0=([PositiveInteger])
#1=(OutputForm) #0#) (#1# $) (NIL))))
12> (HPUT #<hash-table 0000000001ab1f00> (coerce (OutputForm) (#0=([PositiveInteger])
(0#) (#1# $) (NIL)))
(1 (#0# #1=(OutputForm) #0#) (#1# $) (NIL)))
<12 (HPUT (1 (#0=([PositiveInteger])
#1=(OutputForm) #0#) (#1# $) (NIL)))
12> (evalDomain (PositiveInteger))
13> (GETL |print| |TimeTotal|)
```
<13 (GETL 0.0)
13> (GETL |gc| |TimeTotal|)
<13 (GETL 0.0)
13> (PUT |gc| |TimeTotal| 0.0)
<13 (PUT 0.0)
13> (PUT |print| |TimeTotal| 0.0)
<13 (PUT 0.0)
13> (|mkEvalable| ([PositiveInteger]))
14> (CANFUNCALL? [PositiveInteger])
<14 (CANFUNCALL? T)
14> (GETDATABASE [PositiveInteger] CONSTRUCTORKIND)
<14 (GETDATABASE [domain])
14> (GETDATABASE [PositiveInteger] COSIG)
<14 (GETDATABASE (NIL))
<13 (|mkEvalable| ([PositiveInteger]))
13> (GETL [PositiveInteger] LOADED)
<13 (GETL NIL)
13> (|loadLib| [PositiveInteger])
14> (GETL [instantiation] |TimeTotal|)
<14 (GETL 0.0)
14> (GETL |gc| |TimeTotal|)
<14 (GETL 0.0)
14> (PUT |gc| |TimeTotal| 0.0)
<14 (PUT 0.0)
14> (PUT [instantiation] |TimeTotal| 0.0)
<14 (PUT 0.0)
14> (GETDATABASE [PositiveInteger] OBJECT)
<14 (GETDATABASE "/home/daly/noise/mnt/ubuntu/algebra/PI.o")
14> ([pathnameDirectory]
15> (|pathname|)
15> (|pathname|)
15> (#p"/home/daly/noise/mnt/ubuntu/algebra/PI.o")
<14 ([pathnameDirectory]
14> (|pathname|)
<14 ([isSystemDirectory]
14> (|isSystemDirectory| T)
14> ([loadLibNoUpdate] [PositiveInteger] [PositiveInteger]
15> (GETDATABASE [PositiveInteger] CONSTRUCTORKIND)
<15 (GETDATABASE [domain])
15> ([getProplist] [NonNegativeInteger]
15> (((|Category|) (|modemap| (((|Category|) (|Category|) (T *))))
15> (|Join| (|modemap| (((|Category|) (|Category|) (|Category|) (|Category|)))
15> (|Category|))
(T *))

((((Source) (Category) (List (Category)))
  (Category)) (T *)))))
16> (search |NonNegativeInteger|
  ((((Source) (modemap)
    (((Category) (Category)) (T *))))
  (Join) (modemap)
    (((Category) (Category) (Category)) (T *)
      (Category))
    (T *))
  (Category) (Category) (List (Category)))
  (Category)) (T *)))))
17> (searchCurrentEnv |NonNegativeInteger|
  ((((Source) (modemap)
    (((Category) (Category)) (T *))))
  (Join) (modemap)
    (((Category) (Category) (Category)) (Category))
    (Category))
    (T *)
  (Category) (Category) (List (Category)))
  (Category)) (T *)))))
<17 (searchCurrentEnv NIL)
17> (searchTailEnv |NonNegativeInteger| NIL)
<17 (searchTailEnv NIL)
<16 (search| NIL)
16> (search |NonNegativeInteger|
  ((((Source) (modemap)
    (((Category) (Category)) (T *))))
  (Join) (modemap)
    (((Category) (Category) (Category)) (Category))
    (Category))
    (T *)
  (Category) (Category) (List (Category)))
  (Category)) (T *)))))
17> (searchCurrentEnv |NonNegativeInteger|
  ((((Source) (modemap)
    (((Category) (Category)) (T *))))
  (Join) (modemap)
    (((Category) (Category) (Category)) (Category))
    (Category))
    (T *)
    (((Category) (Category) (List (Category)))
      (Category)) (T *)))))
<17 (searchCurrentEnv NIL)
17> (searchTailEnv |NonNegativeInteger| NIL)
<17 (searchTailEnv NIL)
<16 (search| NIL)
<15 (getProplist| NIL)
15> (addBinding |NonNegativeInteger|
  ((SubDomain)
    (PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
\[
(((((\text{Category}) (\text{modemap}) (\text{Category}) (T \ast))) )
((\text{Join}) (\text{modemap})
(((((\text{Category}) (\text{Category}) (\text{Category}) (\text{Category}) (T \ast))) )
(((\text{Category}) (\text{Category}) (\text{List} (\text{Category}) (\text{Category})) (T \ast))))
))
16> (\text{getPropList} (\text{NonNegativeInteger})
(((((\text{Category}) (\text{modemap}) (\text{Category}) (\text{Category})) (T \ast))) )
((\text{Join}) (\text{modemap})
(((((\text{Category}) (\text{Category}) (\text{Category}) (\text{Category}) (\text{Category})) (T \ast))) )
(((\text{Category}) (\text{Category}) (\text{List} (\text{Category}) (\text{Category})) (T \ast))))
))
17> (\text{search} (\text{NonNegativeInteger})
(((((\text{Category}) (\text{modemap}) (\text{Category}) (\text{Category})) (T \ast))) )
((\text{Join}) (\text{modemap})
(((((\text{Category}) (\text{Category}) (\text{Category}) (\text{Category}) (\text{Category})) (T \ast))) )
(((\text{Category}) (\text{Category}) (\text{List} (\text{Category}) (\text{Category})) (T \ast))))
))
18> (\text{searchCurrentEnv} (\text{NonNegativeInteger})
(((((\text{Category}) (\text{modemap}) (\text{Category}) (\text{Category})) (T \ast))) )
((\text{Join}) (\text{modemap})
(((((\text{Category}) (\text{Category}) (\text{Category}) (\text{Category}) (\text{Category})) (T \ast))) )
(((\text{Category}) (\text{Category}) (\text{List} (\text{Category}) (\text{Category})) (T \ast))))
))
<18 (\text{searchCurrentEnv} \text{NIL})
18> (\text{searchTailEnv} (\text{NonNegativeInteger}) \text{NIL})
<18 (\text{searchTailEnv} \text{NIL})
<17 (\text{search} \text{NIL})
17> (\text{search} (\text{NonNegativeInteger})
(((((\text{Category}) (\text{modemap}) (\text{Category}) (\text{Category})) (T \ast))) )
((\text{Join}) (\text{modemap})
(((((\text{Category}) (\text{Category}) (\text{Category}) (\text{Category}) (\text{Category})) (T \ast))) )
(((\text{Category}) (\text{Category}) (\text{List} (\text{Category}) (\text{Category})) (T \ast))))
))
18> (\text{searchCurrentEnv} (\text{NonNegativeInteger})
(((((\text{Category}) (\text{modemap}) (\text{Category}) (\text{Category})) (T \ast))) )
((\text{Join}) (\text{modemap})
(((((\text{Category}) (\text{Category}) (\text{Category}) (\text{Category}) (\text{Category})) (T \ast))) )
(((\text{Category}) (\text{Category}) (\text{List} (\text{Category}) (\text{Category})) (T \ast))))
))
9.2. PARSING THE INPUT

\[
\text{(|Category|)} \text{ (T *)})\)
\]

<18 (|searchCurrentEnv| NIL)
18> (|searchTailEnv| |NonNegativeInteger| NIL)
<18 (|searchTailEnv| NIL)
<17 (|search| NIL)
<16 (|getProplist| NIL)
16> (|addBindingInteractive| |NonNegativeInteger|
\[
\text{(|SubDomain|)
\text{(|PositiveInteger|} \text{ SPADCALL 0 [#1| (QREFELT $ 7)})
\text{(|Category|) (|modemap|
\text{(|Category|) (|Category|) (T *)})
\text{(|Join| (|modemap|
\text{(|Category|) (|Category|) (|Category|)
\text{(|Category|) (T *)})
\text{(|Category|) (|Category|) (|Category|) (T *)})
\text{(|Category|) (|Category|) (|List| (|Category|))
\text{(|Category|) (T *)})})})})
\]

<16 (|addBindingInteractive|)

<15 (|addBinding|
\[
\text{(|NonNegativeInteger|)
\text{(|SubDomain|)
\text{(|PositiveInteger|} \text{ SPADCALL 0 [#1| (QREFELT $ 7)})
\text{(|Category|) (|modemap|
\text{(|Category|) (|Category|) (T *)})
\text{(|Join| (|modemap|
\text{(|Category|) (|Category|) (|Category|)
\text{(|Category|) (T *)})
\text{(|Category|) (|Category|) (|Category|) (T *)})})})})})
\]

15> (|getProplist| |PositiveInteger|

<15 (|addBinding|
\[
\text{(|NonNegativeInteger|)
\text{(|SubDomain|)
\text{(|PositiveInteger|} \text{ SPADCALL 0 [#1| (QREFELT $ 7)})
\text{(|Category|) (|modemap|
\text{(|Category|) (|Category|) (T *)})
\text{(|Join| (|modemap|
\text{(|Category|) (|Category|) (|Category|)
\text{(|Category|) (T *)})
\text{(|Category|) (|Category|) (|Category|) (T *)})})})})})
\]

16> (|search| |PositiveInteger|

<16 (|addBindingInteractive| |NonNegativeInteger|
\[
\text{(|SubDomain|)
\text{(|PositiveInteger|} \text{ SPADCALL 0 [#1| (QREFELT $ 7)})
\text{(|Category|) (|modemap|
\text{(|Category|) (|Category|) (T *)})
\text{(|Join| (|modemap|
\text{(|Category|) (|Category|) (|Category|)
\text{(|Category|) (T *)})
\text{(|Category|) (|Category|) (|List| (|Category|))
\text{(|Category|) (T *)})})})})
\]

<16 (|addBindingInteractive|)

<15 (|addBinding|
\[
\text{(|NonNegativeInteger|)
\text{(|SubDomain|)
\text{(|PositiveInteger|} \text{ SPADCALL 0 [#1| (QREFELT $ 7)})
\text{(|Category|) (|modemap|
\text{(|Category|) (|Category|) (T *)})
\text{(|Join| (|modemap|
\text{(|Category|) (|Category|) (|Category|)
\text{(|Category|) (T *)})
\text{(|Category|) (|Category|) (|Category|) (T *)})})})})})
\]

15> (|getProplist| |PositiveInteger|

<15 (|addBinding|
\[
\text{(|NonNegativeInteger|)
\text{(|SubDomain|)
\text{(|PositiveInteger|} \text{ SPADCALL 0 [#1| (QREFELT $ 7)})
\text{(|Category|) (|modemap|
\text{(|Category|) (|Category|) (T *)})
\text{(|Join| (|modemap|
\text{(|Category|) (|Category|) (|Category|)
\text{(|Category|) (T *)})
\text{(|Category|) (|Category|) (|Category|) (T *)})})})})
\]

16> (|search| |PositiveInteger|
CHAPTER 9. STARTING AXIOM

(((|NonNegativeInteger| |SubDomain|  
  (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7)))  
  |Category| (|modemap|  
  (((|Category|) |Category|) (T *))))  
  |Join| (|modemap|  
  (((|Category|) |Category| |Category|) (T *)))  
  |Category| (|modemap|  
  (((|Category|) |Category| |Category|) (T *))  
  (((|Category|) |Category| |List| |Category|)  
  (|Category|) (T *)))))

17> (|searchCurrentEnv| |PositiveInteger|  
  (((|NonNegativeInteger| |SubDomain|  
  (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7)))  
  |Category| (|modemap|  
  (((|Category|) |Category|) (T *))))  
  |Join| (|modemap|  
  (((|Category|) |Category| |Category|) (T *)))  
  |Category| (|modemap|  
  (((|Category|) |Category| |Category|) (T *))  
  (((|Category|) |Category| |List| |Category|)  
  (|Category|) (T *)))))

<17 (|searchCurrentEnv| NIL)
17> (|searchTailEnv| |PositiveInteger| NIL)
<17 (|searchTailEnv| NIL)

<16 (|search| NIL)
16> (|search| |PositiveInteger|  
  (((|NonNegativeInteger| |SubDomain|  
  (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7)))  
  |Category| (|modemap|  
  (((|Category|) |Category|) (T *))))  
  |Join| (|modemap|  
  (((|Category|) |Category| |Category|) (T *)))  
  |Category| (|modemap|  
  (((|Category|) |Category| |Category|) (T *)))  
  (((|Category|) |Category| |List| |Category|)  
  (|Category|) (T *)))))

17> (|searchCurrentEnv| |PositiveInteger|  
  (((|NonNegativeInteger| |SubDomain|  
  (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7)))  
  |Category| (|modemap|  
  (((|Category|) |Category|) (T *))))  
  |Join| (|modemap|  
  (((|Category|) |Category| |Category|) (T *)))  
  |Category| (|modemap|  
  (((|Category|) |Category| |Category|) (T *)))  
  (((|Category|) |Category| |List| |Category|)  
  (|Category|) (T *)))))

<17 (|searchCurrentEnv| NIL)
17> (|searchTailEnv| |PositiveInteger| NIL)
<17 (|searchTailEnv| NIL)
<16 (|search| NIL)
<15 (|getProplist| NIL)
15> (|addBinding| |PositiveInteger|
   (((|SuperDomain| |NonNegativeInteger|))
    (((((|NonNegativeInteger|)
        (|SubDomain|
         (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
        (|Category| (|modemap|
         (((|Category|) (|Category|)) (T *))))
        (|Join| (|modemap|
         (((|Category|) (|Category|) (|Category|)
             (|Category|)) (T *))
         (((|Category|) (|Category|) (|List| (|Category|))
             (|Category|)) (T *)))))
16> (|getProplist| |PositiveInteger|
   (((((|NonNegativeInteger|)
       (|SubDomain|
        (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
        (|Category| (|modemap|
         (((|Category|) (|Category|)) (T *))))
        (|Join| (|modemap|
         (((|Category|) (|Category|) (|Category|)
             (|Category|)) (T *))
         (((|Category|) (|Category|) (|List| (|Category|))
             (|Category|)) (T *)))))
17> (|search| |PositiveInteger|
   (((((|NonNegativeInteger|)
       (|SubDomain|
        (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
        (|Category| (|modemap|
         (((|Category|) (|Category|)) (T *))))
        (|Join| (|modemap|
         (((|Category|) (|Category|) (|Category|)
             (|Category|)) (T *))
         (((|Category|) (|Category|) (|List| (|Category|))
             (|Category|)) (T *)))))
18> (|searchCurrentEnv| |PositiveInteger|
   (((((|NonNegativeInteger|)
       (|SubDomain|
        (|PositiveInteger| SPADCALL 0 |#1| (QREFELT $ 7))))
        (|Category| (|modemap|
         (((|Category|) (|Category|)) (T *))))
        (|Join| (|modemap|
         (((|Category|) (|Category|) (|Category|)
             (|Category|)) (T *))
         (((|Category|) (|Category|) (|List| (|Category|))
             (|Category|)) (T *)))))
<18 (|searchCurrentEnv| NIL)
18> (|searchTailEnv| |PositiveInteger| NIL)
<18 (|searchTailEnv| NIL)
9.2. PARSING THE INPUT

```
(... ((|Category|) (|Category|) (|List| (|Category|)))
  ((|Category|) (T *)))))))

<15 (|addBinding|
  ( (((|PositiveInteger|)
      (|SuperDomain| |NonNegativeInteger|))
    (|NonNegativeInteger|)
    (|SubDomain|
      (|PositiveInteger| SPADCALL 0 [#1| (QREFELT $ 7)])
    )
  )
  (|modemap|
    (((|Category|) (|Category|)) (T *)))
)

<15 (|Join| (|modemap|
    (((|Category|) (|Category|) (|Category|
        (|Category|)) (T *)))
    (((|Category|) (|Category|) (|List| (|Category|)
        (|Category|)) (T *))))))

15> (|makeByteWordVec2| 1 (0 0 0 0 0 0))
<15 (|makeByteWordVec2| #<bit-vector 0000000001ab1db0>)
15> (|makeByteWordVec2| 12 (2 5 6 0 0 7 2 0 6 0 1 0 0)
  1 0 9 0 1 1 0 6 0 1 2 0 0 0 0 1 2 0 0 0 0 1 1 0 1 1
  0 1 1 0 1 0 1 2 0 0 0 0 1 1 0 1 2 0 1 2 0 0 0 8 1 2
  0 0 0 5 1 0 0 0 1 2 0 6 0 1 2 0 6 0 1 2 0 6 0 0
  1 2 0 6 0 1 2 0 6 0 1 2 0 0 0 1 2 0 0 0 8 1 2
  0 0 0 5 1 2 0 0 0 0 1 2 0 0 8 0 1))
<15 (|makeByteWordVec2| #<vector 0000000001ab1d80>)
15> (GETDATABASE |PositiveInteger| CONSTRUCTORKIND)
<15 (GETDATABASE |domain|)
15> (GETL |load| |TimeTotal|)
<15 (GETL 0.0)
15> (GETL |gc| |TimeTotal|)
<15 (GETL 0.0)
15> (PUT |gc| |TimeTotal| 0.0)
<15 (PUT 0.0)
15> (PUT |load| |TimeTotal| 0.0)
<15 (PUT 0.0)
<14 (|loadLibNoUpdate| T)
<13 (|loadLib| T)
13> (HPUT #<hash-table 000000000105e810> |PositiveInteger|
    ((NIL 1 . #<vector 0000000001ab1d50>)))
<13 (HPUT ((NIL 1 . #<vector 0000000001ab1d50>)))
13> (GETDATABASE |PositiveInteger| CONSTRUCTORKIND)
<13 (GETDATABASE |domain|)
13> (GETL |PositiveInteger| |infovec|)
<13 (GETL (#<vector 0000000000fa5db0>
    #<vector 0000000000fa5cf0>
    (((|commutative| "*" ) . 0))
    (#<bit-vector 0000000001ab1db0>)
    #<vector 0000000000fa5c90>
    #<vector 0000000001ab1de0>) |lookupComplete|)
```
\[ 13> (\text{HPUT} \text{ #<hash-table 000000000105e810> |PositiveInteger|} \\
\quad ((\text{NIL 1 . #<vector 0000000001ab1d50>>)\))
\]
\[ <13 (\text{HPUT} ((\text{NIL 1 . #<vector 0000000001ab1d50>]))) \]
\[ 13> (\text{GETL} \text{|instantiation| |TimeTotal|}) \]
\[ <13 (\text{GETL} 0.0) \]
\[ 13> (\text{GETL} \text{|gc| |TimeTotal|}) \]
\[ <13 (\text{GETL} 0.0) \]
\[ 13> (\text{PUT} \text{|gc| |TimeTotal| 0.0}) \]
\[ <13 (\text{PUT} 0.0) \]
\[ 13> (\text{PUT} \text{|instantiation| |TimeTotal| 0.0}) \]
\[ <13 (\text{PUT} 0.0) \]
\[ <12 (\text{|evalDomain| #<vector 0000000001ab1d50>}) \]
\[ 12> (\text{|compiledLookup| |coerce| ((|OutputForm|) $) \\
\quad \text{#<vector 0000000001ab1d50>}) \]
\[ 13> (\text{|NRTevalDomain| #<vector 0000000001ab1d50>}) \]
\[ 14> (\text{|evalDomain| #<vector 0000000001ab1d50>}) \]
\[ 15> (\text{GETL} \text{|print| |TimeTotal|}) \]
\[ <15 (\text{GETL} 0.0) \]
\[ 15> (\text{GETL} \text{|gc| |TimeTotal|}) \]
\[ <15 (\text{GETL} 0.0) \]
\[ 15> (\text{PUT} \text{|gc| |TimeTotal| 0.0}) \]
\[ <15 (\text{PUT} 0.0) \]
\[ 15> (\text{PUT} \text{|print| |TimeTotal| 0.0}) \]
\[ <15 (\text{PUT} 0.0) \]
\[ 15> (\text{|mkEvalable| #<vector 0000000001ab1d50>}) \]
\[ <15 (\text{|mkEvalable| #<vector 0000000001ab1d50>}) \]
\[ 15> (\text{GETL} \text{|instantiation| |TimeTotal|}) \]
\[ <15 (\text{GETL} 0.0) \]
\[ 15> (\text{GETL} \text{|gc| |TimeTotal|}) \]
\[ <15 (\text{GETL} 0.0) \]
\[ 15> (\text{PUT} \text{|gc| |TimeTotal| 0.0}) \]
\[ <15 (\text{PUT} 0.0) \]
\[ 15> (\text{PUT} \text{|instantiation| |TimeTotal| 0.0}) \]
\[ <15 (\text{PUT} 0.0) \]
\[ <14 (\text{|evalDomain| #<vector 0000000001ab1d50>}) \]
\[ 13> (\text{|basicLookup| |coerce| ((|OutputForm|) $) \\
\quad \text{#<vector 0000000001ab1d50> \#<vector 0000000001ab1d50>}) \]
\[ 14> (\text{|oldCompLookup| |coerce| ((|OutputForm|) $) \\
\quad \text{#<vector 0000000001ab1d50> \#<vector 0000000001ab1d50>}) \]
\[ 15> (\text{|lookupInDomainVector| |coerce| ((|OutputForm|) $) \\
\quad \text{#<vector 0000000001ab1d50> \#<vector 0000000001ab1d50>}) \]
\[ 16> (\text{GETDATABASE} \text{|OutputForm| COSIG}) \]
\[ <16 (\text{GETDATABASE} (\text{NIL})) \]
\[ 16> (\text{GETDATABASE} \text{|PositiveInteger| CONSTRUCTORKIND}) \]
\[ <16 (\text{GETDATABASE} \text{|domain|}) \]
\[ 16> (\text{GETL} \text{|NonNegativeInteger| LOADED}) \]
\[ <16 (\text{GETL} \text{NIL}) \]
\[ <16 (\text{|loadLib| \text{|NonNegativeInteger|}}) \]
\[ 17> (\text{GETL} \text{|print| |TimeTotal|}) \]
9.2. PARSING THE INPUT

<17 (GETL 0.0)
17> (GETL |gc| |TimeTotal|)
17> (GETL 0.0)
17> (PUT |gc| |TimeTotal| 0.0)
17> (PUT 0.0)
17> (PUT |print| |TimeTotal| 0.0)
17> (PUT 0.0)
17> (GETDATABASE |NonNegativeInteger| OBJECT)
17> (GETDATABASE
    "'/home/daly/noise/mnt/ubuntu/algebra/NNI.o")
17> (pathnameDirectory|
    "'/home/daly/noise/mnt/ubuntu/algebra/"
18> (pathname|
    "'/home/daly/noise/mnt/ubuntu/algebra/"
18> (pathname|
    #p"'/home/daly/noise/mnt/ubuntu/algebra/"
17> (pathnameDirectory|
    "'/home/daly/noise/mnt/ubuntu/algebra/"
17> (isSystemDirectory|
    "'/home/daly/noise/mnt/ubuntu/algebra/"
17> (isSystemDirectory| T)
17> (loadLibNoUpdate| |NonNegativeInteger| |
    "'/home/daly/noise/mnt/ubuntu/algebra/"
18> (GETDATABASE |NonNegativeInteger| CONSTRUCTORKIND)
18> (GETDATABASE |domain|)
18> (getProplist| |Integer| ((NIL)))
19> (search| |Integer| ((NIL)))
20> (searchCurrentEnv| |Integer| (NIL))
20> (searchTailEnv| |Integer| NIL)
20> (searchTailEnv| NIL)
20> (search| NIL)
19> (search| |Integer| ((NIL)))
20> (searchCurrentEnv| |Integer| (NIL))
20> (searchCurrentEnv| NIL)
20> (searchTailEnv| |Integer| NIL)
20> (searchTailEnv| NIL)
19> (search| NIL)
18> (getProplist| NIL)
18> (addBinding| |Integer| ((|SubDomain|
    (|NonNegativeInteger| |
    COND
        ((SPADCALL [#1| 0 (QREFELT $ 7)) (QUOTE NIL))
        ((QUOTE T) (QUOTE NIL))) (NIL)))
19> (getProplist| |Integer| ((NIL)))
20> (search| |Integer| ((NIL)))
21> (searchCurrentEnv| |Integer| (NIL))
21> (searchCurrentEnv| NIL)
21> (|searchTailEnv| |Integer| NIL)
<21 (|searchTailEnv| NIL)
<20 (|search| NIL)
20> (|search| |Integer| ((NIL)))
21> (|searchCurrentEnv| |Integer| (NIL))
<21 (|searchCurrentEnv| NIL)
21> (|searchTailEnv| |Integer| NIL)
<21 (|searchTailEnv| NIL)
<20 (|search| NIL)
<19 (|getProplist| NIL)
19> (|addBindingInteractive| |Integer|
   ((((|SubDomain|
       (|NonNegativeInteger|
       COND
       ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
        ((QUOTE T) (QUOTE T)))))) ((NIL)))
<19 (|addBindingInteractive|
   ((((|SubDomain|
       (|NonNegativeInteger|
       COND
       ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
        ((QUOTE T) (QUOTE T)))))))))
<18 (|addBinding|
   ((((|SubDomain|
       (|NonNegativeInteger|
       COND
       ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
        ((QUOTE T) (QUOTE T)))))))))
18> (|getProplist| |NonNegativeInteger|
   ((((|SubDomain|
       (|NonNegativeInteger|
       COND
       ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
        ((QUOTE T) (QUOTE T)))))))))
19> (|search| |NonNegativeInteger|
   ((((|SubDomain|
       (|NonNegativeInteger|
       COND
       ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
        ((QUOTE T) (QUOTE T)))))))))
20> (|searchCurrentEnv| |NonNegativeInteger|
   ((((|SubDomain|
       (|NonNegativeInteger|
       COND
       ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
        ((QUOTE T) (QUOTE T)))))))))
9.2. PARSING THE INPUT

((QUOTE T) (QUOTE T)))

<20 (|searchCurrentEnv| NIL)
20> (|searchTailEnv| |NonNegativeInteger| NIL)
<20 (|searchTailEnv| NIL)
<19 (|search| NIL)
19> (|search| |NonNegativeInteger|

(((|Integer|)
  (|SubDomain|
    (|NonNegativeInteger|
      COND
        ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
        ((QUOTE T) (QUOTE T))))))))))

20> (|searchCurrentEnv| |NonNegativeInteger|

(((|Integer|)
  (|SubDomain|
    (|NonNegativeInteger|
      COND
        ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
        ((QUOTE T) (QUOTE T))))))))

<20 (|searchCurrentEnv| NIL)
20> (|searchTailEnv| |NonNegativeInteger| NIL)
<20 (|searchTailEnv| NIL)
<19 (|search| NIL)
18> (|getProplist| NIL)
18> (|getProplist| |NonNegativeInteger|

(((|SuperDomain| |Integer|))
  (((|Integer|)
      (|SubDomain|
        (|NonNegativeInteger|
          COND
            ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
            ((QUOTE T) (QUOTE T))))))))

19> (|getProplist| |NonNegativeInteger|

(((|Integer|)
  (|SubDomain|
    (|NonNegativeInteger|
      COND
        ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
        ((QUOTE T) (QUOTE T))))))))

20> (|search| |NonNegativeInteger|

(((|Integer|)
  (|SubDomain|
    (|NonNegativeInteger|
      COND
        ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
        ((QUOTE T) (QUOTE T))))))))

21> (|searchCurrentEnv| |NonNegativeInteger|

(((|Integer|)
  (|SubDomain|
    (|NonNegativeInteger|
      COND
        ((SPADCALL |#1| 0 (QREFELT $ 7)) (QUOTE NIL))
        ((QUOTE T) (QUOTE T))))))))
CHAPTER 9. STARTING AXIOM

COND
((SPADCALL [#1| 0 (QREFELT $ 7)) (QUOTE NIL))
((QUOTE T) (QUOTE T))))))))
<21 (|searchCurrentEnv| NIL)
21> (|searchTailEnv| |NonNegativeInteger| NIL)
<21 (|searchTailEnv| NIL)
<20 (|search| NIL)
20> (|search| |NonNegativeInteger|
((|Integer|
  (|SubDomain|
    (|NonNegativeInteger|
      COND
      ((SPADCALL [#1| 0 (QREFELT $ 7)) (QUOTE NIL))
       ((QUOTE T) (QUOTE T))))))))
21> (|searchCurrentEnv| |NonNegativeInteger|
((|Integer|
  (|SubDomain|
    (|NonNegativeInteger|
      COND
      ((SPADCALL [#1| 0 (QREFELT $ 7)) (QUOTE NIL))
       ((QUOTE T) (QUOTE T))))))))
<21 (|searchCurrentEnv| NIL)
21> (|searchTailEnv| |NonNegativeInteger| NIL)
<21 (|searchTailEnv| NIL)
<20 (|search| NIL)
<19 (|getProplist| NIL)
19> (|addBindingInteractive| |NonNegativeInteger|
((|SuperDomain| |Integer|))
((|Integer|
  (|SubDomain|
    (|NonNegativeInteger|
      COND
      ((SPADCALL [#1| 0 (QREFELT $ 7)) (QUOTE NIL))
       ((QUOTE T) (QUOTE T))))))))
<19 (|addBindingInteractive|
((|(NonNegativeInteger| (|SuperDomain| |Integer|))
 |Integer|
  (|SubDomain|
    (|NonNegativeInteger|
      COND
      ((SPADCALL [#1| 0 (QREFELT $ 7)) (QUOTE NIL))
       ((QUOTE T) (QUOTE T))))))))
<18 (|addBinding|
((|(NonNegativeInteger| (|SuperDomain| |Integer|))
 |Integer|
  (|SubDomain|
    (|NonNegativeInteger|
      COND
      ((SPADCALL [#1| 0 (QREFELT $ 7)) (QUOTE NIL))
       ((QUOTE T) (QUOTE T))))))))
9.2. Parsing the Input

18> (|makeByteWordVec2| 1 (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
<18 (|makeByteWordVec2| #<bit-vector 0000000001ab1990>)

18> (|makeByteWordVec2| 18 (2 5 6 0 0 7 2 5 0 0 10 2 0 6
0 0 1 1 0 6 0 1 2 0 0 0 0 8 2 0 1 1 0 0 12 2 0 0 0 5
9 0 0 0 1 2 0 0 0 0 1 1 0 1 1 0 1 1 0 0 0 1 2 0 0 0
0 1 1 0 6 0 1 2 0 0 0 0 1 2 0 0 0 0 1 1 0 1 7 0 1 1
0 1 6 0 1 2 0 0 0 0 1 2 0 1 1 0 1 2 0 1 3 0 0 1 1 0
18 0 1 2 0 0 0 14 1 2 0 0 0 15 1 0 0 1 0 0 0 1 2
0 6 0 0 1 2 0 6 0 0 1 2 0 6 0 0 1 2 0 6 0 0 1 2 0
6 0 0 1 2 0 0 0 0 1 2 0 0 0 14 1 2 0 0 0 15 1 2 0
0 0 0 1 2 0 0 14 0 1 2 0 0 15 0 1))
<18 (|makeByteWordVec2| #<vector 0000000001ab1960>)

18> (GETDATABASE |NonNegativeInteger| CONSTRUCTORKIND)
<18 (GETDATABASE |domain|)
18> (GETL |load| |TimeTotal|)
<18 (GETL 0.0)
18> (GETL |gc| |TimeTotal|)
<18 (GETL 0.0)
18> (PUT |gc| |TimeTotal| 0.0)
<18 (PUT 0.0)
18> (PUT |load| |TimeTotal| 0.0)
<18 (PUT 0.0)
17> (|loadLibNoUpdate| T)
<16 (|loadLib| T)
16> (HPUT #<hash-table 000000000105e810> |NonNegativeInteger|
   ((NIL 1 . #<vector 0000000001ab1930>)))
<16 (HPUT ((NIL 1 . #<vector 0000000001ab1930>)))
16> (GETDATABASE |NonNegativeInteger| CONSTRUCTORKIND)
<16 (GETDATABASE |domain|)
16> (GETL |NonNegativeInteger| |infovec|)
<16 (GETL (|#<vector 0000000001ab1b10>
   #<vector 0000000001ab1ae0>
   #<vector 0000000001ab1ae0>
   ( (((|commutative| "*") . 0))
   #<bit-vector 0000000001ab1990>
   #<vector 0000000001ab16b0>
   #<vector 0000000001ab1a80>
   . #<vector 0000000001ab1960>)
   |lookupComplete|))
16> (HPUT #<hash-table 000000000105e810> |NonNegativeInteger|
   ((NIL 1 . #<vector 0000000001ab1930>)))
<16 (HPUT ((NIL 1 . #<vector 0000000001ab1930>)))
16> (|lookupInDomainVector| |coerce| (((|OutputForm|) $)
   #<vector 0000000001ab1930> #<vector 0000000001ab1d50>)
   |lookupComplete|))
17> (GETDATABASE |NonNegativeInteger| CONSTRUCTORKIND)
<17 (GETDATABASE |domain|)
17> (PNAME |NonNegativeInteger|)
<17 (PNAME "NonNegativeInteger")
17> (PNAME |NonNegativeInteger|)
CHAPTER 9. STARTING AXIOM

<17 (PNAME "NonNegativeInteger")
17> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
17> (GETDATABASE (NIL))
17> (GETDATABASE |PositiveInteger| CONSTRUCTORKIND)
17> (GETDATABASE |domain|)
17> (GETL |Integer| LOADED)
17> (GETL NIL)
17> ([loadLib] |Integer|)
18> (GETL |print| |TimeTotal|)
18> (GETL 0.0)
18> (GETL |gc| |TimeTotal|)
18> (GETL 0.0)
18> (PUT |gc| |TimeTotal| 0.0)
18> (PUT 0.0)
18> (PUT |print| |TimeTotal| 0.0)
18> (PUT 0.0)
18> (GETDATABASE |Integer| OBJECT)
18> (GETDATABASE "/home/daly/noise/mnt/ubuntu/algebra/INT.o")
18> ([pathnameDirectory] "/home/daly/noise/mnt/ubuntu/algebra/INT.o")
18> ([pathname] "'/home/daly/noise/mnt/ubuntu/algebra/INT.o")
19> ([pathname] #p"'/home/daly/noise/mnt/ubuntu/algebra/INT.o")
18> ([pathnameDirectory] "'/home/daly/noise/mnt/ubuntu/algebra/")
18> ([isSystemDirectory] "'/home/daly/noise/mnt/ubuntu/algebra/")
18> ([isSystemDirectory] T)
18> ([loadLibraryNoUpdate] |Integer| |Integer|
"'/home/daly/noise/mnt/ubuntu/algebra/INT.o")
19> (GETDATABASE |Integer| CONSTRUCTORKIND)
19> (GETDATABASE |domain|)
19> ([makeByteWordVec2] 1
(0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
<19 ([makeByteWordVec2] #<bit-vector 00000000001723f60>)
19> ([makeByteWordVec2] 133
(1 7 6 0 8 3 7 6 0 9 9 10 2 7 6 0 11 12 1 7 6 0 13 0
14 0 15 2 7 0 9 16 1 7 6 0 17 1 7 6 0 18 1 7 6 0
19 1 35 0 11 36 1 44 0 11 45 1 47 0 11 48 1 50 0 11
51 1 9 0 11 53 2 93 90 91 92 94 1 97 95 96 98 1 96
0 0 99 1 96 2 0 100 1 101 95 96 102 1 96 0 2 103 1
0 104 0 105 2 108 95 96 107 109 2 110 95 96 111
1 101 95 96 112 1 96 21 0 113 1 96 0 0 114 1 116 96
115 117 2 0 21 0 0 1 1 0 21 0 25 1 0 87 0 88 1 0 0
0 89 1 0 21 0 1 2 0 0 0 1 2 0 0 83 0 1 3 0 0 0 0
0 42 1 0 0 0 1 1 0 104 0 1 2 0 21 0 0 1 1 0 11 0 1
2 0 0 0 0 82 0 0 0 1 1 0 124 0 1 1 0 11 0 1 2 0 0 0
9.2. PARSING THE INPUT

0 81 2 0 60 58 61 62 0 57 58 59 1 0 83 0 85 1 0
120 0 1 1 0 21 0 1 1 0 121 0 1 1 0 0 65 0 0 0 64
2 0 0 0 0 80 1 0 127 126 1 1 0 21 0 1 3 0 0 0 0 1
2 0 0 0 0 56 1 0 21 0 1 2 0 0 0 0 1 3 0 122 0 123
122 1 1 0 21 0 26 1 0 21 0 75 1 0 83 0 1 1 0 21 0
34 2 0 125 126 0 1 3 0 0 0 0 43 2 0 0 0 77 2 0
0 0 0 76 1 0 0 1 1 0 0 40 2 0 131 0 0 1 1 0 0
126 1 2 0 0 0 0 1 1 0 9 0 55 2 0 0 0 0 1 0 0 1 1
0 0 0 31 1 0 0 33 1 0 133 0 1 2 0 118 118 118 119
2 0 0 0 86 1 0 0 126 1 1 0 0 0 1 1 0 104 0 105 3
0 129 0 0 0 1 2 0 130 0 0 1 2 0 83 0 0 84 2 0 125
126 0 1 1 0 21 0 1 1 0 73 0 1 2 0 78 0 0 79 1 0 0
1 2 0 0 0 73 1 1 0 0 0 32 1 0 0 30 1 0 9 0 54 1 0
47 0 49 1 0 44 0 46 1 0 50 0 52 1 0 123 0 1 1 0 11
0 39 1 0 0 11 38 1 0 0 0 1 1 0 11 38 1 0 35 0 37
0 0 73 1 2 0 21 0 1 2 0 0 0 1 0 0 0 29 2 0 21 0
0 1 3 0 0 0 0 41 1 0 0 63 2 0 0 0 73 1 2 0 0 0
132 1 0 0 0 27 0 0 0 28 3 0 6 7 0 21 24 2 0 9 0 21
22 2 0 6 7 0 23 1 0 9 0 20 1 0 0 1 2 0 0 0 73 1 2
0 21 0 1 2 0 21 0 0 1 2 0 21 0 0 66 2 0 21 0 0 1
2 0 21 0 0 67 2 0 0 0 70 1 0 0 68 2 0 0 0 69
2 0 0 73 74 2 0 0 0 132 1 2 0 0 0 71 2 0 0 11 0
72 2 0 0 73 0 1 2 0 0 132 0 1)
<19 (makeByteWordVec2 |vector 000000001723f00>)
19> (GETDATABASE |Integer| CONSTRUCTORKIND)
<19 (GETDATABASE |domain|)
19> (GETL |load| |TimeTotal|)
<19 (GETL 0.0)
19> (GETL |gc| |TimeTotal|)
<19 (GETL 0.0)
19> (PUT |gc| |TimeTotal| 0.0)
<19 (PUT 0.0)
19> (PUT |load| |TimeTotal| 0.0)
<19 (PUT 0.0)
<18 (loadLibNoUpdate| T)
<17 (loadLib| T)
17> (HPUT #<hash-table 00000000001ab1750> |Integer|
   ((NIL 1 . #<vector 000000001723ed0>))
<17 (HPUT ((NIL 1 . #<vector 000000001723ed0>)))
17> (GETDATABASE |Integer| CONSTRUCTORKIND)
<17 (GETDATABASE |domain|)
17> (GETL |Integer| |infovec|)
<17 (GETL (#vector 00000000001ab1780>
   (#vector 00000000001ab1750> (|infinite| . 0)
   (|noetherian| . 0) (|canonicalsClosed| . 0)
   (|canonical| . 0) (|canonicalUnitNormal| . 0)
   (|multiplicativeValuation| . 0)
   (|noZeroDivisors| . 0)
   (|commutative| "*" . 0) (|rightUnitary| . 0)
   (|leftUnitary| . 0) (|unitsKnown| . 0))

CHAPTER 9. STARTING AXIOM

(#<bit-vector 0000000001723f60>
  #<vector 0000000001723fc0>
  #<vector 0000000001723f90>
  . #<vector 0000000001723f00>)

[lookupComplete])

17> (HPUT #<hash-table 000000000105e810> |Integer|
    ((NIL 1 . #<vector 0000000001723ed0>)))

17> (HPUT ((NIL 1 . #<vector 0000000001723ed0>)))

17> (lookupInDomainVector| coerce| ((|OutputForm|) $)
    #<vector 0000000001723ed0> #<vector 0000000001ab1d50>)

18> (GETDATABASE |Integer| CONSTRUCTORKIND)

18> (GETDATABASE |domain|)

18> (PNAME |Integer|)

18> (PNAME "Integer")

18> (PNAME |Integer|)

18> (PNAME "Integer")

18> (GETDATABASE |OutputForm| COSIG)

18> (GETDATABASE (NIL))

18> (GETDATABASE |PositiveInteger| CONSTRUCTORKIND)

18> (GETDATABASE |domain|)

17> (lookupInDomainVector|
    (#<compiled-function |INT;coerce;$Of;16|>
     . #<vector 0000000001723ed0>))

16> (lookupInDomainVector|
    (#<compiled-function |INT;coerce;$Of;16|>
     . #<vector 0000000001723ed0>))

15> (lookupInDomainVector|
    (#<compiled-function |INT;coerce;$Of;16|>
     . #<vector 0000000001723ed0>))

14> (oldCompLookup|
    (#<compiled-function |INT;coerce;$Of;16|>
     . #<vector 0000000001723ed0>))

13> (basicLookup|
    (#<compiled-function |INT;coerce;$Of;16|>
     . #<vector 0000000001723ed0>))

12> (compiledLookup|
    (#<compiled-function |INT;coerce;$Of;16|>
     . #<vector 0000000001723ed0>))

"TPD:INT:coerce(x):OutputForm"

12> (GETDATABASE |Integer| CONSTRUCTORKIND)

12> (GETDATABASE |domain|)

12> (GETL |OutputForm| LOADED)

12> (GETL NIL)

12> (loadLib |OutputForm|)

13> (GETL |print| |TimeTotal|)

13> (GETL 0.0)

13> (GETL |gc| |TimeTotal|)

13> (GETL 0.0)
9.2. PARSING THE INPUT

13> (PUT |gc| |TimeTotal| 0.0)
<13 (PUT 0.0)
13> (PUT |print| |TimeTotal| 0.0)
<13 (PUT 0.0)
13> (GETDATABASE |OutputForm| OBJECT)
<13 (GETDATABASE
  "/home/daly/noise/mnt/ubuntu/algebra/OUTFORM.o")
13> (|pathnameDirectory|
  "/home/daly/noise/mnt/ubuntu/algebra")
14> (|pathname|
  "/home/daly/noise/mnt/ubuntu/algebra/OUTFORM.o")
13> (|pathname| #p"/home/daly/noise/mnt/ubuntu/algebra/OUTFORM.o")
<13 (|pathnameDirectory|
  "/home/daly/noise/mnt/ubuntu/algebra/")
13> (|isSystemDirectory| T)
13> (|loadLibNoUpdate| |OutputForm| |OutputForm|
  "/home/daly/noise/mnt/ubuntu/algebra/OUTFORM.o")
<14 (|loadLibNoUpdate| |OutputForm| |OutputForm|
  "/home/daly/noise/mnt/ubuntu/algebra/OUTFORM.o")
14> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<14 (GETDATABASE |domain|)
14> (|makeByteWordVec2| 1 (0 0))
14> (|makeByteWordVec2| #<bit-vector 0000000001723c90>)
14> (|makeByteWordVec2| 144
  (1 10 9 0 11 0 25 0 26 2 10 0 0 25 27 2 10 0 0 25 0 28
   2 19 0 0 0 36 2 19 0 0 0 37 2 19 9 0 0 46 1 6 0 0 56
   2 6 0 0 0 57 1 6 9 0 69 1 6 0 0 70 1 6 2 0 71 1 6 73
   0 74 1 19 9 0 75 2 76 0 0 0 77 1 76 0 0 105 1 25 0
  10 114 2 10 0 73 25 115 1 73 9 0 128 2 73 9 0 129
  1 131 10 130 132 1 10 0 0 133 2 0 9 0 0 1 2 0 0 0
  120 0 0 0 19 35 1 0 19 0 30 1 0 0 19 47 1 0 0 52 80 2
   0 0 0 0 48 2 0 0 0 52 78 1 0 19 0 33 2 0 0 0 66 2
   0 0 0 0 136 3 0 0 0 0 137 1 0 0 0 135 1 0 19 0 32
   2 0 0 0 0 65 1 0 0 109 2 0 0 0 124 1 0 0 52 55
   2 0 0 0 52 72 2 0 0 19 19 49 1 0 0 121 2 0 0 0
  122 1 0 0 0 45 2 0 0 0 19 42 2 0 0 0 93 2 0 0 0
  127 1 0 0 0 110 2 0 0 0 94 3 0 0 0 0 140 1 0 0
  0 138 2 0 0 0 139 1 0 7 0 8 2 0 0 0 73 117 1 0 0
  0 113 2 0 0 0 68 2 0 0 0 67 2 0 0 0 52 104 2 0
  0 0 0 108 1 0 0 52 53 1 0 0 52 64 1 0 0 63 2 0
  0 0 118 1 0 0 0 111 2 0 0 0 123 1 0 0 10 29 1 0
  23 24 1 0 0 21 22 1 0 0 19 20 2 0 0 0 97 1 0 0
  98 1 0 7 10 14 1 0 0 10 13 1 0 0 50 51 1 0 0 44 2
  0 0 0 19 41 1 0 10 0 1 2 0 0 0 126 3 0 0 0
  0 0 0 0 143 2 0 0 0 0 142 1 0 0 0 141 1 0 0 0 141 1 0 0 102 2 0 0
  52 106 3 0 0 0 0 107 1 0 0 19 38 0 0 19 34 1 0 19
  0 31 1 0 0 52 79 2 0 0 0 39 1 0 144 0 1 2 0 0 0
  95 0 0 0 12 2 0 0 0 52 103 2 0 0 0 73 116 1 0 0
  112 2 0 0 0 92 2 0 0 0 73 134 1 0 0 52 54 1 0 17

CHAPTER 9. STARTING AXIOM

0 18 1 0 0 0 43 2 0 0 0 19 40 1 0 0 0 61 1 0 0 52
62 1 0 0 52 60 1 0 0 0 59 1 0 0 119 1 0 0 52 58 2
0 0 0 0 101 2 0 0 0 125 2 0 0 0 96 2 0 0 0 81
1 0 0 0 100 2 0 0 0 99 2 0 0 0 85 2 0 0 0 83
2 0 0 0 0 16 2 0 0 0 15 2 0 0 0 84 2 0 0 0 82
2 0 0 0 0 90 1 0 0 0 88 2 0 0 0 87 2 0 0 0 86 2
0 0 0 0 91 2 0 0 0 0 89))
<14 (\makeByteWordVec2\ #<vector 0000000001723c60>)
14> (GETDATABASE \OutputForm\ CONSTRUCTORKIND)
<14 (GETDATABASE \domain\)
14> (GETL \load\ \TimeTotal\)
<14 (GETL 0.0)
14> (GETL \gc\ \TimeTotal\)
<14 (GETL 0.0)
14> (PUT \gc\ \TimeTotal\ 0.0)
<14 (PUT 0.0)
14> (PUT \load\ \TimeTotal\ 0.0)
<14 (PUT 0.0)
<13 (\loadLib\ |loadLibNoUpdate| T)
<12 (\loadLib\ T)
12> (HPUT #<hash-table 0000000000105e810 \OutputForm\ ((NIL 1 . \#<vector 0000000001723c30>))
<12 (HPUT ((NIL 1 . \#<vector 0000000001723c30>)))
12> (GETDATABASE \OutputForm\ CONSTRUCTORKIND)
<12 (GETDATABASE \domain\)
12> (GETL \OutputForm\ \infovec\)
<12 (GETL (#<vector 0000000001723d80>
    #<vector 0000000001723d50> NIL
    (#<bit-vector 0000000001723c90>
        #<vector 0000000001723cf0>
        #<vector 0000000001723cc0>
        . #<vector 0000000001723c60>)
    \lookupComplete\)))
12> (GETL \List\ \LOADED\)
<12 (GETL NIL)
12> (\loadLib\ \List\)
13> (GETL \print\ \TimeTotal\)
<13 (GETL 0.0)
13> (GETL \gc\ \TimeTotal\)
<13 (GETL 0.0)
13> (PUT \gc\ \TimeTotal\ 0.0)
<13 (PUT 0.0)
13> (PUT \print\ \TimeTotal\ 0.0)
<13 (PUT 0.0)
13> (GETDATABASE \List\ OBJECT)
<13 (GETDATABASE
    "/home/daly/noise/mnt/ubuntu/algebra/LIST.o")
<13 (\pathnameDirectory\ 
    "/home/daly/noise/mnt/ubuntu/algebra/LIST.o")
14> (\pathname|
9.2. PARSING THE INPUT

"/home/daly/noise/mnt/ubuntu/algebra/LIST.o")
<14 (|pathname| #p"/home/daly/noise/mnt/ubuntu/algebra/LIST.o")
<13 (|pathnameDirectory|
  "/home/daly/noise/mnt/ubuntu/algebra/")
13> (|isSystemDirectory|
  "/home/daly/noise/mnt/ubuntu/algebra/")
<13 (|isSystemDirectory| T)
13> (|loadLibNoUpdate| |List| |List|
  "/home/daly/noise/mnt/ubuntu/algebra/LIST.o")
14> (GETDATABASE |List| CONSTRUCTORKIND)
<14 (GETDATABASE |domain|)
14> (|makeByteWordVec2| 8
  (0 0 0 0 0 0 0 0 3 0 8 4 0 8 1 2 4 5))
<14 (|makeByteWordVec2| #<vector 0000000001723a20>)
14> (GETDATABASE |List| CONSTRUCTORKIND)
<14 (GETDATABASE |domain|)
14> (|makeByteWordVec2| 51
  (1 13 12 0 14 3 13 12 0 15 15 16 1 0 6 0 17 3 6 12
   13 0 8 18 1 0 0 0 19 1 13 12 0 20 0 21 0 22 2 13 0
   15 21 23 1 13 12 0 24 1 13 12 0 25 1 13 12 0 26 1
   0 15 0 27 2 0 15 0 8 28 2 0 12 13 0 29 3 0 12 13 0
   8 30 2 0 0 0 0 31 1 0 0 0 32 2 0 0 0 0 33 0 0 0 34
   1 0 8 0 35 2 0 8 6 0 36 2 0 0 0 0 37 2 0 6 0 38 39
   2 0 0 0 40 2 0 0 0 0 41 1 42 0 15 43 1 44 0 42 45
   1 6 44 0 46 2 47 0 44 0 48 1 44 0 49 50 1 0 44 0 51
   2 1 0 0 0 33 2 1 0 0 0 37 2 1 0 0 0 41 1 0 0 0 19 1
   1 0 32 1 0 8 0 9 0 0 0 7 2 1 8 6 0 36 1 0 6 0 17
   1 0 8 0 35 0 0 0 34 2 0 6 0 38 39 1 2 44 0 51 2 0 0
   6 0 10 2 0 0 6 0 40 2 0 0 0 0 31 2 0 0 0 0 11 3 5 12
   13 0 8 30 2 5 12 13 0 29 1 5 15 0 27 2 5 15 0 8 28))
14> (|makeByteWordVec2| #<vector 0000000001723a50>)
14> (GETDATABASE |List| CONSTRUCTORKIND)
<14 (GETDATABASE |domain|)
14> (GETL |load| |TimeTotal|)
<14 (GETL 0.0)
14> (GETL |gc| |TimeTotal|)
<14 (GETL 0.0)
14> (PUT |gc| |TimeTotal| 0.0)
<14 (PUT 0.0)
14> (PUT |load| |TimeTotal| 0.0)
<14 (PUT 0.0)
<13 (|loadLibNoUpdate| T)
<12 (|loadLib| T)
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
CHAPTER 9. STARTING AXIOM

12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (GETDATABASE |SetCategory| COSIG)
<12 (GETDATABASE (NIL))
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (GETDATABASE |SetCategory| COSIG)
<12 (GETDATABASE (NIL))
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (GETDATABASE |Integer| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (PNAME |Integer|)
<12 (PNAME "Integer")
12> (PNAME |Integer|)
<12 (PNAME "Integer")
12> (GETDATABASE |OrderedSet| COSIG)
<12 (GETDATABASE (NIL))
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (GETDATABASE (NIL))
12> (GETDATABASE (NIL))
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
9.2. PARSING THE INPUT

12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (PNAME |OutputForm|)
<12 (PNAME "OutputForm")
12> (HPUT #<hash-table 000000000105e810> |List|
(((|OutputForm|) 1 . #<vector 00000000017239f0>))
<12 (HPUT (((|OutputForm|) 1
 . #<vector 00000000017239f0>)))
12> (GETDATABASE |List| CONSTRUCTORKIND)
<12 (GETDATABASE |domain|)
12> (GETL |List| |infovec|)
<12 (GETL (#<vector 0000000001723b10>
|lookupIncomplete|))
12> (HPUT #<hash-table 000000000105e810> |OutputForm|
((NIL 1 . #<vector 0000000001723c30>))
<12 (HPUT ((NIL 1 . #<vector 0000000001723c30>)))
12> (GETDATABASE |Integer| COSIG)
<12 (GETDATABASE (NIL))
12> (|basicLookup| |outputForm| ($ (|Integer|))
#<vector 0000000001723c30> #<vector 0000000001723c30>)
13> (|oldCompLookup| |outputForm| ($ (|Integer|))
#<vector 0000000001723c30> #<vector 0000000001723c30>)
14> (|lookupInDomainVector| |outputForm| ($ (|Integer|))
#<vector 0000000001723c30> #<vector 0000000001723c30>)
15> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<15 (GETDATABASE |domain|)
15> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<15 (GETDATABASE |domain|)
15> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<15 (GETDATABASE |domain|)
15> (GETDATABASE |OutputForm| CONSTRUCTORKIND)
<15 (GETDATABASE |domain|)
15> (GETDATABASE |Integer| COSIG)
<15 (GETDATABASE (NIL))
14> (|lookupInDomainVector|)
"TPD::OUTFORM::outputForm n"

12> (GETL |print| |TimeTotal|)
12> (GETL 0.0)
12> (GETL |gc| |TimeTotal|)
12> (GETL 0.0)
12> (PUT |gc| |TimeTotal| 0.0)
12> (PUT 0.0)
12> (PUT |print| |TimeTotal| 0.0)
12> (PUT 0.0)
12> (|member| 1 ("failed" "nil" "prime" "sqfr" "irred"))
12> (|member| NIL)
12> (|member| EQUATNUM (SLASH OVER))
12> (|member| NIL)
12> (GETL EQUATNUM |Led|)
12> (GETL ([dummy] [dummy] 10000 0))
12> (|member| EQUATNUM (SLASH OVER))
12> (|member| NIL)
12> (GETL EQUATNUM |Led|)
12> (GETL ([dummy] [dummy] 10000 0))
12> (GETL EQUATNUM INFIXOP)
12> (GETL " ")
12> (GETL EQUATNUM WIDTH)
12> (GETL NIL)
12> (GETL EQUATNUM APP)
12> (GETL NIL)
12> (|member| EQUATNUM (SLASH OVER))
12> (|member| NIL)
12> (GETL EQUATNUM |Led|)
12> (GETL ([dummy] [dummy] 10000 0))
12> (|member| EQUATNUM (SLASH OVER))
12> (|member| NIL)
12> (GETL EQUATNUM |Led|)
12> (GETL ([dummy] [dummy] 10000 0))
12> (GETL EQUATNUM INFIXOP)
12> (GETL " ")
12> (GETL EQUATNUM SUPERSPAN)
12> (GETL NIL)
12> (GETL EQUATNUM SUBSPAN)
12> (GETL NIL)

(1) 1
9.2. PARSING THE INPUT

12> ((|putHist| % |value| (??PosInt? 1) (NIL)))
13> ((|recordNewValue| % |value| (??PosInt? 1)))
14> ((GETL |print| |TimeTotal|)
<14 (GETL 0.0)
14> ((GETL |gc| |TimeTotal|)
<14 (GETL 0.0)
14> ((PUT |gc| |TimeTotal| 0.0)
<14 (PUT 0.0)
14> ((PUT |print| |TimeTotal| 0.0)
<14 (PUT 0.0)
14> ((|recordNewValue0| % |value| (??PosInt? 1)))
<14 (|recordNewValue0|
    (?? (|value| (??PosInt? 1)))))
14> ((GETL |history| |TimeTotal|)
<14 (GETL 0.0)
14> ((GETL |gc| |TimeTotal|)
<14 (GETL 0.0)
14> ((PUT |gc| |TimeTotal| 0.0)
<14 (PUT 0.0)
14> ((PUT |history| |TimeTotal| 0.0)
<14 (PUT 0.0)
13> ((|recordNewValue| |history|)
13> ((|search| % (NIL)))
14> ((|searchCurrentEnv| % (NIL))
<14 (|searchCurrentEnv| NIL)
14> ((|searchTailEnv| % NIL)
<14 (|searchTailEnv| NIL)
13> ((|search| NIL)
12> ((|putHist| (?? (|value| (??PosInt? 1))))))
12> ((|printTypeAndTime| 1 (??PosInt?))
14> ((|sayKeyedMsg| "%rjon Type: %1p %rjoff" (??PosInt?)))
15> ((|sayKeyedMsgLocal| "%rjon Type: %1p %rjoff" (??PosInt?)))
16> ((|segmentKeyedMsg| "%rjon %1p %rjoff")
<16 (|segmentKeyedMsg| ("%rjon" "Type:" "%1p" "%rjoff"))
16> ((|member| "%rjon" (??COn| "%rj"))
16> ((|member| "%rjon")
16> ((|member| "%type")
<16 (|member| NIL)
16> ((|member| "%1p" (??COff| "%ceoff" "%rjoff" "%rjoff"))
16> ((|member| "%1p")
16> ((|member| "%rjoff")
<16 (|member| NIL)
16> ((|member| "%rjoff")
<16 (|member| NIL)
16> ((|member| "%rj")
<16 (|member| NIL)
16> ((|member| "%rj")
<16 (|member| NIL)
16> ((|member| "%rj")
<16 (|member| NIL)
CHAPTER 9. STARTING AXIOM

16> (|member| "Type:" (|%ceon| "%ceon"))
<16 (|member| NIL)
16> (|member| "Type:" (|%rjon| "%rjon"))
<16 (|member| NIL)
16> (|member| "%1p" (|%ceon| "%ceon"))
<16 (|member| NIL)
16> (|member| "%1p" (|%rjon| "%rjon"))
<16 (|member| NIL)
16> (DIGITP #\r)
<16 (DIGITP NIL)
16> (DIGITP #\1)
<16 (DIGITP 1)
16> (GETDATABASE |PositiveInteger| ABBREVIATION)
<16 (GETDATABASE PI)
16> (|member| "Type:" (|%n| "%n"))
<16 (|member| NIL)
16> (|member| "Type:" (|%y| "%y"))
<16 (|member| NIL)
16> (|member| "%rj" (|%ce| "%ce" |%rj| "%rj" "%rj"))
<16 (|member| NIL)
16> (|member| "%rj" (|%n| "%n" |%n|))
<16 (|member| NIL)
16> (|member| "%rj" (|%y| "%y" |%y|))
<16 (|member| NIL)
16> (SIZE "Type:")
<16 (SIZE 5)
16> (|member| |PositiveInteger| "Type:")
<16 (|member| NIL)
16> (|sayMSG| ("%rj" "Type:" |PositiveInteger|))
17> (SAYBRIGHTLY1 ("%rj" "Type:" |PositiveInteger|))
18> (BRIGHTPRINT ("%rj" "Type:" |PositiveInteger|))
19> (|member| "%rj" (|PositiveInteger| "%rj" "%rj"))
<19 (|member| NIL)
19> (|member| "Type:" (|%l| "%l"))
<19 (|member| NIL)
9.2. PARSING THE INPUT

19> (|member| " " (|%l| "%l"))
<19 (|member| NIL)
19> (|member| |PositiveInteger| (|%l| "%l"))
<19 (|member| NIL)
19> (|member| "Type:" ("%b" "%d" |%b| |%d|))
<19 (|member| NIL)
19> (|member| "Type:" ("%l" |%l|))
<19 (|member| NIL)
19> (|member| = " ("%b" "%d" |%b| |%d|))
<19 (|member| NIL)
19> (|member| = " ("%l" |%l|))
<19 (|member| NIL)
19> (|member| |PositiveInteger| ("%b" "%d" |%b| |%d|))
<19 (|member| NIL)
19> (|member| |PositiveInteger| ("%l" |%l|))
<19 (|member| NIL)
19> (PNAME |PositiveInteger|)
<19 (PNAME "PositiveInteger")

PositiveInteger

18 (BRIGHTPRINT NIL)
<17 (SAYBRIGHTLY1 NIL)
<16 (sayMSG NIL)
<15 (sayKeyDownedMsgLocal NIL)
<14 (sayKeyDownedMsg NIL)
<12 (printTypeAndTime NIL)
<11 (recordAndPrint |done|)
<11 (recordFrame |normal|)
11> (|diffAlist|
   ((% (|value| (|PositiveInteger|) . 1))) NIL)
<12 (|diffAlist| ((% (|value|))))
<11 (recordFrame ((% (|value|))))
11> (GETL |print| |TimeTotal|)
<11 (GETL 0.0)
11> (GETL |gc| |TimeTotal|)
<11 (GETL 0.0)
11> (PUT |gc| |TimeTotal| 0.0)
<11 (PUT 0.0)
11> (PUT |print| |TimeTotal| 0.0)
<11 (PUT 0.0)
10 (|processInteractive| ((|PositiveInteger|) . 1))
10> (|writeHistModesAndValues|)
11> (|putHist| % |value|
   #0=((|PositiveInteger|) . 1) (((% (|value| . #0#)))))
CHAPTER 9. STARTING AXIOM

```lisp
12> (recordNewValue % value ((PositiveInteger) . 1))
13> (GETL other | TimeTotal|)
<13 (GETL 0.0)
13> (GETL gc | TimeTotal|)
<13 (GETL 0.0)
13> (PUT gc | TimeTotal| 0.0)
<13 (PUT 0.0)
13> (PUT other | TimeTotal| 0.0)
<13 (PUT 0.0)
13> (recordNewValue0 % value ((PositiveInteger) . 1))
<13 (recordNewValue0 (value (PositiveInteger) . 1))
13> (GETL history | TimeTotal|)
<13 (GETL 0.0)
13> (GETL gc | TimeTotal|)
<13 (GETL 0.0)
13> (PUT gc | TimeTotal| 0.0)
<13 (PUT 0.0)
13> (PUT history | TimeTotal| 0.0)
<13 (PUT 0.0)
<12 (recordNewValue | history|)
12> (search % (((% (value (PositiveInteger) . 1)))))
13> (searchCurrentEnv % (((% (value (PositiveInteger) . 1)))))
<13 (searchCurrentEnv (((value (PositiveInteger) . 1))))
12 (search (((value (PositiveInteger) . 1))))
<11 (putHist (((% (value (PositiveInteger) . 1))))))
<10 (writeHistModesAndValues NIL)
10> (updateHist)
11> (GETL other | TimeTotal|)
<11 (GETL 0.0)
11> (GETL gc | TimeTotal|)
<11 (GETL 0.0)
11> (PUT gc | TimeTotal| 0.0)
<11 (PUT 0.0)
11> (PUT other | TimeTotal| 0.0)
<11 (PUT 0.0)
11> (updateInCoreHist)
<11 (updateInCoreHist 1)
11> (writeHiFi)
<11 (writeHiFi

((1 (% (value (PositiveInteger) . 1))))))
11> (disableHist)
<11 (disableHist NIL)
11> (updateCurrentInterpreterFrame)
12> (createCurrentInterpreterFrame)
<12 (createCurrentInterpreterFrame

(frame0

((((% (value . #0=(PositiveInteger) . 1)))))))
```
9.2. PARSING THE INPUT

2 T
#1=(NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL . #1#)
20 1 NIL ((1 (% (% (1 (value) . #0#)))
#<vector 0000000000fa5cc0>)
12> (|updateFromCurrentInterpreterFrame|)
<12 (|updateFromCurrentInterpreterFrame| NIL)
<11 (|updateCurrentInterpreterFrame| NIL)
11> (GETL |history| |TimeTotal|)
<11 (GETL 0.0)
11> (GETL |gc| |TimeTotal|)
<11 (GETL 0.0)
11> (PUT |gc| |TimeTotal| 0.0)
<11 (PUT 0.0)
11> (PUT |history| |TimeTotal| 0.0)
<11 (PUT 0.0)
10> (|updateHist| |history|)
9 ((|processInteractive| ((|PositiveInteger|) . 1))
<8 (|intInterpretPform| ((|PositiveInteger|) . 1))
8> (|ncPutQ|
((|carrier| (|ok?| . T)
(|ptreePremacro|
 . #0=([|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
 . "1")
 (|ptree| . #0#)
(|lines| ((#1# . 1) . "1")
(|messages|)
(|stepNumber| . 1))) |value|
((|PositiveInteger|) . 1))
9> (|ncAlist|
((|carrier| (|ok?| . T)
(|ptreePremacro|
 . #0=([|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
 . "1")
 (|ptree| . #0#)
(|lines| ((#1# . 1) . "1")
(|messages|)
(|stepNumber| . 1))))
<9 (|ncAlist| ((|ok?| . T)
(|ptreePremacro|
 . #0=([|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
 . "1")
 (|ptree| . #0#)
(|lines| ((#1# . 1) . "1")
(|messages|)
(|stepNumber| . 1))))
9> (|ncAlist|
((|carrier| (|ok?| . T)
(|ptreePremacro|
 . #0=([|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
 . "1")
 (|ptree| . #0#)
(|lines| ((#1# . 1) . "1")
(|messages|)
(|stepNumber| . 1))))
CHAPTER 9. STARTING AXIOM

. "1")
(\ptree . #0#)
(\lines ((#1# . 1) . "1")
(\messages)
(\stepNumber . 1))

\ncalist
(((\ok? . T)
  (\ptreepremacro
   . #0=((\integer (\posn #1=(0 "1" 1 1 "strings") . 0)
     . "1"))
  (\ptree . #0#)
  (\lines ((#1# . 1) . "1")
  (\messages)
  (\stepNumber . 1)))

\ncTag
(((\carrier (\ok? . T)
  (\ptreepremacro
   . #0=((\integer (\posn #1=(0 "1" 1 1 "strings") . 0)
     . "1"))
  (\ptree . #0#)
  (\lines ((#1# . 1) . "1")
  (\messages)
  (\stepNumber . 1)))))

\ncTag (\carrier)
\ncPutQ ((\PositiveInteger) . 1)
\nphinterpret ((\PositiveInteger) . 1))

\ncConversationPhase,wrapup
(((\carrier ((\value (\PositiveInteger) . 1) (\ok? . T)
  (\ptreepremacro
   . #0=((\integer (\posn #1=(0 "1" 1 1 "strings") . 0)
     . "1"))
  (\ptree . #0#)
  (\lines ((#1# . 1) . "1")
  (\messages)
  (\stepNumber . 1))))

\ncConversationPhase,wrapup| NIL)
\ncEltQ ((\carrier ((\value (\PositiveInteger) . 1) (\ok? . T)
  (\ptreepremacro
   . #0=((\integer (\posn #1=(0 "1" 1 1 "strings") . 0)
     . "1"))
  (\ptree . #0#)
  (\lines ((#1# . 1) . "1")
  (\messages)
  (\stepNumber . 1)))

\ncalist
(((\carrier ((\value (\PositiveInteger) . 1) (\ok? . T)
  (\ptreepremacro
   . "1")
  (\ptree . #0#)
  (\lines ((#1# . 1) . "1")
  (\messages)
  (\stepNumber . 1)))))
9.2. PARSING THE INPUT

```lisp
. #0=(((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
  . "1")
  (%ptree . #0#)
  (%lines (((#1# . 1) . "1")
  (%messages)
  (%stepNumber . 1)))))
<7 (%ncAlist|
  (%value ((|PositiveInteger|) . 1) (%ok? . T)
  (%ptreePremacro)
  . #0=(((|integer| (|posn| #1=(0 "1" 1 1 "strings") . 0))
  . "1")
  (%ptree . #0#)
  (%lines (((#1# . 1) . "1")
  (%messages)
  (%stepNumber . 1)))))
<6 (%ncEltQ| NIL)
<5 (%intloopSpadProcess,interp| NIL)
<4 (%intloopSpadProcess| 2)

4> (%StreamNull|
  (%nonnullstream| #0=|incAppend1| NIL
  (%nonnullstream| #2=|next1| |ncloopParse|
  (%nonnullstream| #0# NIL
  (%nonnullstream| #2# |lineoftoks| (%nullstream))))))
5> (%incAppend1| NIL
  (%nonnullstream| #0=|next1| |ncloopParse|
  (%nonnullstream| |incAppend1| NIL
  (%nonnullstream| #0# |lineoftoks| (%nullstream))))))
6> (%StreamNull| NIL)
<6 (%StreamNull| T)
6> (%StreamNull|
  (%nonnullstream| #0=|next1| |ncloopParse|
  (%nonnullstream| |incAppend1| NIL
  (%nonnullstream| #0# |lineoftoks| (%nullstream))))))
7> (%next1| |ncloopParse|
  (%nonnullstream| |incAppend1| NIL
  (%nonnullstream| |next1| |lineoftoks| (%nullstream))))
8> (%StreamNull|
  (%nonnullstream| |incAppend1| NIL
  (%nonnullstream| |next1| |lineoftoks| (%nullstream))))
9> (%incAppend1| NIL
  (%nonnullstream| |next1| |lineoftoks| (%nullstream))))
10> (%StreamNull| NIL)
<10 (%StreamNull| T)
10> (%StreamNull|
  (%nonnullstream| |next1| |lineoftoks| (%nullstream))))
11> (%next1| |lineoftoks| (%nullstream))
12> (%StreamNull| (%nullstream))
<12 (%StreamNull| T)
<11 (%next1| (%nullstream))
```
<10 (StreamNull T)
<9 (incAppend1 (nullstream))
<8 (StreamNull T)
<7 (next1 (nullstream))
<6 (StreamNull T)
<5 (incAppend1 (nullstream))
<4 (StreamNull T)
<3 (intloopProcess 2)
Chapter 10

Axiom Details

10.1 Variables Used

10.2 Data Structures

10.3 Functions

Set the restart hook

When a lisp image containing code is reloaded there is a hook to allow a function to be called. In our case it is the restart function which is the entry to the Axiom interpreter.

```lisp
(set-restart-hook Void → 'restart)
— defun set-restart-hook 0 —
```

```
(defun set-restart-hook ()
  "Set the restart hook"
  #+KCL (setq system::*top-level-hook* 'restart)
  #+Lucid (setq boot::restart-hook 'restart)
  'restart
)
```

237
restart function (The restart function)

The restart function is the real root of the world. It sets up memory if we are working in a GCL/akcl version of the system.

The `compiler::*compile-verbose*` flag has been set to nil globally. We do not want to know about the microsteps of GCL’s compile facility.

The `compiler::*suppress-compiler-warnings*` flag has been set to t. We do not care that certain generated variables are not used.

The `compiler::*suppress-compiler-notes*` flag has been set to t. We do not care that tail recursion occurs.

It sets the current package to be the “BOOT” package which is the standard package in which the interpreter runs.

The “initroot” (11.3 p 259) function sets global variables that depend on the AXIOM shell
variable. These are needed to find basic files like s2-us.msgs, which contains the error message text.

The “openserver” (31 p 1133) function tried to set up the socket connection used for things like hyperdoc. The \$openServerIfTrue variable starts true, which implies trying to start a server.

Axiom has multiple frames that contain independent information about a computation. There can be several frames at any one time and you can shift back and forth between the frames. By default, the system starts in “frame0” (try the \texttt{frame names} command). See the Frame Mechanism chapter (2.1 page 3).

The “printLoadMsgs” (25.44 p 940) variable controls whether load messages will be output as library routines are loaded. We disable this by default. It can be changed by using \texttt{set message autoload}.

The “current-directory” (11.3 p 266) variable is set to the current directory. This is used by the \texttt{cd} function and some of the compile routines.

The “statisticsInitialization” (35 p 1181) function initializes variables used to collect statistics. Currently, only the garbage collector information is initialized.

We test \texttt{*ThisIsARunningSystem*}(p891). If this variable is true then we are restarting from a previously running system and we do not want to reset all of the user variables.

\begin{verbatim}
[init-memory-config p258]
[initroot p259]
[openserver p1133]
[makeInitialModemapFrame p261]
[get-current-directory p260]
[statisticsInitialization p1181]
[initHist p820]
[initializeInterpreterFrameRing p12]
[spadStartUpMsgs p241]
[restart0 p241]
[readSpadProfileIfThere p1107]
[spad p242]
[\$openServerIfTrue p131]
[\$SpadServerName p133]
[\$SpadServer p133]
[\$I0index p24]
[\$InteractiveFrame p23]
[\$printLoadMsgs p940]
[\$current-directory p206]
[\$displayStartMsgs p953]
[\$currentLine p??]

— defun restart —

(defun restart ()

)
(declare (special $openServerIfTrue $SpadServerName |$SpadServer| |
$IIndex| |$InteractiveFrame| |$printLoadMsgs| $current-directory |
$displayStartMsgs| |$currentLine|))
#:akcl
(init-memory-config :cons 1024 :fixnum 200 :symbol 500 :package 8 |
$array 800 :string 1024 :cpages 6000 :rpages 2000 :hole 4000)
#:akcl (setq compiler::*compile-verbose* nil)
#:akcl (setq compiler::*suppress-compiler-warnings* t)
#:akcl (setq compiler::*suppress-compiler-notes* t)
#:akcl (setq si::*system-directory* "")
(in-package "BOOT")
(initroot)
#:akcl
(unless *ThisIsARunningSystem*
 (setq |$InteractiveFrame| (|makeInitialModemapFrame|))
 (|initHist|)
 (|initializeInterpreterFrameRing|)
 (setq *ThisIsARunningSystem* t)
 (when |$displayStartMsgs| (|spadStartUpMsgs|))
 (setq |$currentLine| nil)
 (restart0)
 (|readSpadProfileIfThere|)
 (|spad|)

defvar localVars
— initvars —
(defvar |$localVars| () ) ;checked by isType
10.3. FUNCTIONS

defun Non-interactive restarts

(defun restart0 ()
  (interpopen) ;; open up the interpreter database
  (operationopen) ;; all of the operations known to the system
  (categoryopen) ;; answer hasCategory question
  (browseopen))

---

defun The startup banner messages

(defun spadStartUpMsgs ()
  (let (bar)
    (declare (special $msgAlist $opSysName $linelength *yearweek* *build-version*)
      (when (> $linelength 60)
        (setq bar (fillerSpaces $linelength (specialChar '|hbar|)))
        (sayKeyedMsg
          (format nil "%ceton AXIOM Computer Algebra System %l Version: %1 %l ~
                Timestamp: %2 %ceoff")
          (list *build-version* *yearweek*)))
    (sayMSG bar)
    (say " Issue )copyright to view copyright notices.")
    (say " Issue )summary for a summary of useful system commands.")
    (say " Issue )quit to leave AXIOM and return to shell.")
    (say " Visit http://axiom-developer.org for more information")
defun Make a vector of filler characters

(defun fillerSpaces (&rest arglist &aux charPart n)
  (setq n (car arglist))
  (setq charPart (cdr arglist))
  (if (<= n 0)
      ""n
    (make-string n :initial-element (character (or (ifcar charPart) " ")))))

---

defvar $PrintCompilerMessageIfTrue

The $PrintCompilerMessageIfTrue variable is set to NIL in spad.

---

Starts the interpreter but do not read in profiles

(setOutputAlgebra p967)
(runspad p243)
[$PrintCompilerMessageIfTrue p242]

---

(defun spad ()
  "Starts the interpreter but do not read in profiles"
  (let (|$PrintCompilerMessageIfTrue|)
    (declare (special |$PrintCompilerMessageIfTrue|))
    (setq |$PrintCompilerMessageIfTrue| nil)
10.3. FUNCTIONS

(|setOutputAlgebra| '|%initialize%|)
(|runspad|)
'|EndOfSpad|)

---

defvar $quitTag

— initvars —

(defvar $quitTag system::*quit-tag*)

---

defun runspad

[quitTag p243]
[coerceFailure p??]
[top-level p??]
[seq p??]
[exit p??]
[resetStackLimits p244]
[ncTopLevel p248]
[$quitTag p243]

— defun runspad —

(defun runspad ()
  (prog (mode)
    (declare (special $quitTag))
    (return
      (seq
        (progn
          (setq mode '|restart|)
          (do ()
            ((null (eq mode '|restart|)) nil)
            (seq
              (exit
                (progn
                  (resetStackLimits))
                (catch $quitTag
                  (catch '|coerceFailure|
                    (catch '|top_level| (ncTopLevel)))))
              (setq mode (catch '|top_level| (ncTopLevel)))))))))))
defun Reset the stack limits

[reset-stack-limits p??]

— defun resetStackLimits 0 —

(defun |resetStackLimits| ()
  "Reset the stack limits"
  (system:reset-stack-limits))
Chapter 11

Handling Terminal Input

11.1 Streams

defvar curinstream

The curinstream variable is set to the value of the *standard-input* common lisp variable in ncIntLoop. While not using the “dollar” convention this variable is still “global”.

— initvars —

(defvar curinstream (make-synonym-stream '*standard-input*))

----------

defvar curoutstream

The curoutstream variable is set to the value of the *standard-output* common lisp variable in ncIntLoop. While not using the “dollar” convention this variable is still “global”.

— initvars —

(defvar curoutstream (make-synonym-stream '*standard-output*))

----------

defvar errorinstream

— initvars —
(defvar errorinstream (make-synonym-stream '*terminal-io*))

---

(defvar erroroutstream

---

(defvar *eof*

---

(defvar *whitespace*

---

There are several different environments used in the interpreter:

$\texttt{InteractiveFrame}$ is the environment where the user values are stored. Any side effects of evaluation of a top-level expression are stored in this environment. It is always used as the starting environment for interpretation.

$\texttt{e}$ is the name used for $\texttt{InteractiveFrame}$ while interpreting.

$\texttt{env}$ is local environment used by the interpreter. Only temporary information (such as types of local variables) is stored in $\texttt{env}$. It is thrown away after evaluation of each expression.
defvar $InteractiveMode

    — initvars —

(defvar |$InteractiveMode| (list (list nil)) "top level environment")

------

defvar $env

    — initvars —

(defvar |$env| nil "checked in isDomainValuedVariable")

------

defvar $e

The $e variable is set to the value of $InteractiveFrame which is set in restart to the value of the call to the makeInitialModemapFrame function. This function simply returns a copy of the variable $InitialModemapFrame.

Prints out the value x which is of type m, and records the changes in environment $e into $InteractiveFrame Thus $e is a copy of the variable $InitialModemapFrame.

This variable is used in the undo mechanism.

    — initvars —

(defvar |$e| nil "the environment?")

------

defvar $InteractiveMode

    — initvars —

(defvar |$InteractiveMode| t)
defvar $boot

— initvars —

(defvar $boot nil)

$newspad

The $newspad is set to T in ncTopLevel.

defvar $newspad

— initvars —

(defvar $newspad nil)

Top-level read-parse-eval-print loop

Top-level read-parse-eval-print loop for the interpreter. Uses the Bill Burge’s parser. [ncInt-
Loop p249]

[$e p247]

[$spad p242]

[$newspad p248]

[$boot p758]

[$InteractiveMode p247]

[$InteractiveFrame p23]

[*eof* p246]

[in-stream p1107]

— defun ncTopLevel —

(defun ncTopLevel ()
"Top-level read-parse-eval-print loop"
(let ([|$e| $spad $newspad $boot |$InteractiveMode| *eof* in-stream])
 (declare (special |$e| $spad $newspad $boot |$InteractiveMode| *eof* in-stream |$InteractiveFrame|))
 (setq in-stream curinstream))
(setq *eof* nil)
(setq |$InteractiveMode| t)
(setq $boot nil)
(setq $newspad t)
(setq $spad t)
(setq |$e| |$InteractiveFrame|
(\texttt{ncIntLoop}\}))

---

defun ncIntLoop

\texttt{[intloop\ p250]}
\texttt{[curinstream\ p245]}
\texttt{[curoutstream\ p245]}

---

defvar $intTopLevel

---

defvar $intRestart

---
defun intloop

Note that the SpadInterpretStream function uses a list of three strings as an argument. The values in the list seem to have no use and can eventually be removed. [intTopLevel p249]  
[SpadInterpretStream p252]  
[resetStackLimits p244]  
[$intTopLevel p249]  
[$intRestart p249]  

— defun intloop —

(defun intloop ()  
(prog (mode)  
 (declare (special $intTopLevel $intRestart))  
 (return  
 (progn  
 (setq mode $intRestart)  
 (lambda ()  
 (loop  
 (cond  
 ((not (equal mode $intRestart))  
 (return nil))  
 (t  
 (progn  
 ($resetStackLimits)  
 (setq mode  
 (catch $intTopLevel  
 ($SpadInterpretStream 1  
 (list ‘tim ‘daly ‘?) t))))))))))))

——

defvar $ncMsgList

— initvars —

(defun $ncMsgList nil)

——

defun SpadInterpretStream

The SpadInterpretStream function takes three arguments
11.1. STREAMS

str This is passed as an argument to intloopReadConsole

source This is the name of a source file but appears not to be used. It is set to the list
(tim daly ?).

interactive? If this is false then various messages are suppressed and input does not
use piles. If this is true then the library loading routines might output messages and
piles are expected on input (as from a file).

System commands are handled by the function in the “hook” variable $\texttt{systemCommandFunction}$
which has the default function \texttt{InterpExecuteSpadSystemCommand}. Thus, when a system com-
mand is entered this function is called.

defun GCL cmpnote function

GCL keeps noting the fact that the compiler is performing tail-recursion. Bill Schelter added
this as a debugging tool for Axiom and it was never removed. Patching the lisp code in the
GCL build fails as the system is actually built from the pre-compiled C code. Thus, we can
only step on this message after the fact. The cmpnote function is used nowhere else in GCL
so stepping on the function call seems best. We’re unhappy with this hack and will try to
convince the GCL crowd to fix this.

— defun cmpnote —

\#:gcl (defun compiler::cmpnote (&rest x) (declare (ignore x)))

---

defvar $newcompErrorCount

— initvars —

(defvar $newcompErrorCount 0)

---

defvar $nopos

— initvars —

(defvar $nopos (list '|noposition|))
[mkprompt p267]
[intloopReadConsole p254]
[intloopInclude p287]
[$systemCommandFunction p??]
[$ncMsgList p250]
[$erMsgToss p??]
[$lastPos p??]
[$inclAssertions p??]
[$okToExecuteMachineCode p??]
[$newcompErrorCount p251]
[$libQuiet p??]
[$fn p??]
[$nopos p251]

--- defun SpadInterpretStream ---

(defun |SpadInterpretStream| (str source interactive?)
  (let (|$systemCommandFunction| |$ncMsgList| |$erMsgToss| |$lastPos| |$inclAssertions|
    |$okToExecuteMachineCode| |$newcompErrorCount|
    |$libQuiet|)
    (declare (special
               |$systemCommandFunction| |$ncMsgList| |$erMsgToss| |$lastPos|
               |$inclAssertions| |$okToExecuteMachineCode| |$newcompErrorCount|
               |$libQuiet| |$nopos|))
  (setq |$libQuiet| (null interactive?))
  (setq |$newcompErrorCount| 0)
  (setq |$okToExecuteMachineCode| t)
  (setq |$inclAssertions| (list 'aix '|CommonLisp|))
  (setq |$lastPos| |$nopos|)
  (setq |$erMsgToss| nil)
  (setq |$ncMsgList| nil)
  (setq |$systemCommandFunction| #'|InterpExecuteSpadSystemCommand|)
  (if interactive?
      (progn
        (princ (mkprompt))
        (|intloopReadConsole| "" str)
        (|intloopInclude| source 0))))
11.2 The Read-Eval-Print Loop

```lisp
defun intloopReadConsole

Note that this function relies on the fact that lisp can do tail-recursion. The function recursively invokes itself.

The serverReadLine function is a special readline function that handles communication with the session manager code, which is a separate process running in parallel.

We read a line from standard input.

- If it is a null line then we exit Axiom.
- If it is a zero length line we prompt and recurse
- If $\texttt{dalymode}$ and open-paren we execute lisp code, prompt and recurse The $\texttt{dalymode}$ will interpret any input that begins with an open-paren as a lisp expression rather than Axiom input. This is useful for debugging purposes when most of the input lines will be lisp. Setting $\texttt{dalymode}$ non-nil will certainly break user expectations and is to be used with caution.
- If it is $\texttt{\textbar tel " or } \texttt{\textbar tel n"$ we drop into lisp. Use the (restart) function to return to the interpreter loop.
- If it starts with $\texttt{\textbar c$ we process the command, prompt, and recurse
- If it is a command then we remember the current line, process the command, prompt, and recurse.
- If the input has a trailing underscore (Axiom line-continuation) then we cut off the continuation character and pass the truncated string to ourselves, prompt, and recurse
- otherwise we process the input, prompt, and recurse.

Notice that all but two paths (a null input or a $\texttt{\textbar fi$ or a $\texttt{\textbar fin$) will end up as a recursive call to ourselves.
```

[top-level p??]
[serverReadLine p269]
[leaveScratchpad p873]
[mkprompt p267]
[intloopReadConsole p254]
[intloopPrefix? p259]
[intmplisp p260]
[setCurrentLine p266]
[ncloopCommand p750]
[concat p1197]
[ncloopEscaped p261]
[intloopProcessString p261]
intloopReadConsole : (String Integer) → Throw
— defun intloopReadConsole —

(defun intloopReadConsole (prefix stepNumber)
  (declare (special $dalymode))
  (let (newStepNo cmd pfx input)
    ; read the next line
    (setq input (serverReadLine *standard-input*))
    ; if we have lost *standard-input* then exit Axiom
    (when (null (stringp input)) (leaveScratchpad)))
    ; if the input is a zero-length input, recurse
    (when (eq (length input) 0)
      (princ (mkprompt))
      (intloopReadConsole "" stepNumber))
    ; if $dalymode is nonnil anything starting with '(' is a lisp expression
    ; evaluate the expression in lisp and recurse
    (when (and $dalymode (intloopPrefix? "(" input))
      (intnplisp input)
      (princ (mkprompt))
      (intloopReadConsole "" stepNumber))
    ; if the input starts with ")fi" or ")fin" throw into lisp
    (setq pfx (intloopPrefix? ")fi" input))
    (when (and pfx (or (string= pfx ")fi") (string= pfx ")fin")))
      (throw '|top_level| nil))
    ; if the input starts with ')' it is a command; execute and recurse
    (when (and (equal prefix ")") (setq cmd (intloopPrefix? ")" input)))
      (setCurrentLine cmd)
      (setq newStepNo (ncloopCommand cmd stepNumber))
      (princ (mkprompt))
      (intloopReadConsole "" newStepNo))
    ; if the last non-blank character on the line is an underscore
    ; we use the current accumulated input as a prefix and recurse.
    ; this has the effect of concatenating the next line (minus the underscore)
    (setq input (concat prefix input))
    (when (ncloopEscaped input)
      (intloopReadConsole (subseq input 0 (- (length input) 1)) stepNumber))
    ; if there are no special cases, process the current line and recurse
    (setq newStepNo (intloopProcessString input stepNumber))
    (princ (mkprompt))
    (intloopReadConsole "" newStepNo)))
11.3 Helper Functions

Get the value of an environment variable

[getenv p??]

(defun getenviron (var)
  "Get the value of an environment variable"
  (defun getenviron (var)
    "Get the value of an environment variable"
    #+allegro (sys::getenv (string var))
    #+clisp (ext:getenv (string var))
    #+(or cmu scl)
      (cdr
        (assoc (string var) ext:*environment-list* :test #'equalp :key #'string))
    #+(or kcl akcl gcl) (si::getenv (string var))
    #+lispworks (lw:environment-variable (string var))
    #+lucid (lcl:environment-variable (string var))
    #+mcl (ccl::getenv var)
    #+sbcl (sb-ext:posix-getenv var)
  )
)

---

defvar $intCoerceFailure

---

(initvars)

(defvar |$intCoerceFailure| 'coerceFailure)

---

defvar $intSpadReader

---

(initvars)

(defvar |$intSpadReader| 'SPAD_READER)
defun InterpExecuteSpadSystemCommand

[|intCoerceFailure p255|]
[|intSpadReader p255|]
[|ExecuteInterpSystemCommand p256|]
[|$intSpadReader p255|]
[|$intCoerceFailure p255|]

— defun InterpExecuteSpadSystemCommand —

(defun |InterpExecuteSpadSystemCommand| (string)
  (declare (special |$intSpadReader| |$intCoerceFailure|))
  (catch |$intCoerceFailure|
    (catch |$intSpadReader|
      (|ExecuteInterpSystemCommand| string)))

——

defun ExecuteInterpSystemCommand

[|intProcessSynonyms p257|]
[|substring p256|]
[|doSystemCommand p718|]
[|$currentLine p??|]

— defun ExecuteInterpSystemCommand —

(defun |ExecuteInterpSystemCommand| (string)
  (let (|$currentLine|)
    (declare (special |$currentLine|))
    (setq string (|intProcessSynonyms| string))
    (setq |$currentLine| string)
    (setq string (substring string 1 nil))
    (unless (equal string "") (|doSystemCommand| string))))

——

defun substring

— defun substring 0 —

(defun substring (cvec start length)
  (if length
(subseq (string cvec) start (+ start length))
(subseq (string cvec) start))

---

defun Handle Synonyms

(defun processSynonyms p257
	(line p??)

---

defun intProcessSynonyms ---

(defun intProcessSynonyms (str)
	(let ((line str))
		(declare (special line))
		(processSynonyms!)
		line))

---

defun Synonym File Reader

(defvar p1196
	(strpos p1196
	
	(substring p256
	
	(string2id-n p??)
	
	(lassoc p??)
	
	(strconc p??)
	
	(size p1196)
	
	(concat p1197)
	
	(rplacstr p??)
	
	(processSynonyms p257)
	
	(\$CommandSynonymAlist p749)
	
	(line p??)

---

defun processSynonyms ---

(defun processSynonyms ()
	(let (fill p aline synstr syn to opt fun cl chr)
		(declare (special \$CommandSynonymAlist line))
		(setq p (strpos "\$" line 0 nil))
		(setq fill "))
		(cond
			(p
				(setq aline (substring line p nil)))
(when (> p 0) (setq fill (substring line 0 p))))
(t
 (setq p 0)
 (setq aline line)))
(setq to (strpos " " aline 1 nil))
(cond (to (setq to (1- to))))
(setq synstr (substring aline 1 to))
(setq syn (string2id-n synstr 1))
(when (setq fun (lassoc syn |$CommandSynonymAlist|))
 (setq to (strpos ")" fun 1 nil))
(cond
 ((and to (not (eql to (1- (size fun))))))
 (setq opt (strconc " " (substring fun to nil)))
 (setq fun (substring fun 0 (1- to )))
 (t (setq opt " ")))
(when (> (size synstr) (size fun))
 (do ((G167173 (size synstr)) (i (size fun) (1+ i)))
     ((> i G167173) nil)
     (setq fun (concat fun " "))))
(setq cl (strconc fill (rplacstr aline 1 (size synstr) fun) opt))
(setq line cl)
(setq chr (elt line (1+ p)))
(|processSynonyms|)))

---

defun init-memory-config

Austin-Kyoto Common Lisp (AKCL), now known as Gnu Common Lisp (GCL) requires
some changes to the default memory setup to run Axiom efficiently. This function performs
those setup commands.

[allocate p??]
[allocate-contiguous-pages p??]
[allocate-relocatable-pages p??]
[set-hole-size p??]

--- defun init-memory-config 0 ---

(defun init-memory-config (&key
 (cons 500)
 (fixnum 200)
 (symbol 500)
 (package 8)
 (array 400)
 (string 500)
 (cfun 100)
 (cpages 3000)
## 11.3. HELPER FUNCTIONS

```lisp
;; initialize AKCL memory allocation parameters
#+:AKCL
(progn
 (system:allocate 'cons cons)
 (system:allocate 'fixnum fixnum)
 (system:allocate 'symbol symbol)
 (system:allocate 'package package)
 (system:allocate 'array array)
 (system:allocate 'string string)
 (system:allocate 'cfun cfun)
 (system:allocate-contiguous-pages cpages)
 (system:allocate-relocatable-pages rpages)
 (system:set-hole-size hole))
#-:AKCL
nil)
```

### Set spadroot to be the AXIOM shell variable

Sets up the system to use the **AXIOM** shell variable if we can and default to the `$spadroot` variable (which was the value of the **AXIOM** shell variable at build time) if we can’t.

```lisp
(defun initroot
 (declare (special $spadroot))
 (reroot (or newroot $spadroot (error "setenv AXIOM or (setq $spadroot)")))
)
```

### Does the string start with this prefix?

If the prefix string is the same as the whole string initial characters -R(ignoring spaces in the whole string) then we return the whole string minus any leading spaces.

```lisp
(defun |intloopPrefix| 0
 (defun |intloopPrefix| (prefix whole)
 )
```
"Does the string start with this prefix?"
(let ((newprefix (string-left-trim '(#\space) prefix))
    (newwhole (string-left-trim '(#\space) whole)))
  (when (<= (length newprefix) (length newwhole))
    (when (string= newprefix newwhole :end2 (length prefix))
      newwhole)))

---

defun Interpret a line of lisp code
This is used to handle )lisp top level commands [nplisp p744]
[currentLine p??]

---

defun intnplisp ---

(defun intnplisp (s)
  (declare (special $currentLine))
  (setq $currentLine s)
  (nplisp $currentLine))

---

Get the current directory

---

defun get-current-directory 0 ---

(defun get-current-directory ()
  "Get the current directory"
  (namestring (truename "")))

---

Prepend the absolute path to a filename
Prefix a filename with the AXIOM shell variable.
[spadroot p133]

---

defun make-absolute-filename 0 ---

(defun make-absolute-filename (name)
11.3. HELPER FUNCTIONS

"Prepend the absolute path to a filename"
(declare (special $spadroot))
(concatenate 'string $spadroot name))

---------------------

Make the initial modemap frame

(defun makeInitialModemapFrame ()
"Make the initial modemap frame"
(declare (special $InitialModemapFrame)))
(copy $InitialModemapFrame))

---------------------

defun ncloopEscaped

The ncloopEscaped function will return true if the last non-blank character of a line is an underscore, the Axiom line-continuation character. Otherwise, it returns nil.

(defun ncloopEscaped (x)
(let ((l (length x)))
(dotimes (i l)
  (when (char= (char x (- l i 1)) #\_) (return t))
  (unless (char= (char x (- l i 1)) #\space) (return nil))))))

---------------------

defun intloopProcessString

(intloopProcessString : (String,StepNo) → StepNo)

(defun intloopProcessString (String,StepNo) → StepNo)

---------------------
(defun |intloopProcessString| (currentline stepno)
  (|setCurrentLine| currentline)
  (|intloopProcess| stepno t
    (|next| #'|ncloopParse|)
    (|next| #'|lineoftoks| (|incString| currentline)))))

---

**defun ncloopParse**

[ncloopDQlines p296]
[npParse p365]
[dqToList p566]

— defun ncloopParse —

(defun |ncloopParse| (s)
  (let (cudr lines stream dq t1)
    (setq t1 (car s))
    (setq dq (car t1))
    (setq stream (cadr t1))
    (setq t1 (|ncloopDQlines| dq stream))
    (setq lines (car t1))
    (setq cudr (cadr t1))
    (cons (list (list lines (|npParse| (|dqToList| dq)))) (cdr s))))

---

**defun next**

[Delay p329]
[next1 p203]

next : (Function,Delay) → Delay

— defun next —

(defun |next| (function delay)
  (|Delay| #'|next1| (list function delay)))
11.3. HELPER FUNCTIONS


defun next1

[StreamNull p555]
[incAppend p312]
[next p262]

next1 : Delay → ParsePair
         — defun next1 —

(defun next1 (&rest delayArg)
  (let (h delay function)
    (setq function (car delayArg))
    (setq delay (cadr delayArg))
    (cond
      ((StreamNull delay) StreamNil)
      (t
        (setq h (apply function (list delay)))
        (incAppend (car h) (next function (cdr h)))))))

defun incString

The incString function gets a string, usually from Axiom's input, and constructs a set of nested function calls to process the input line. [incRenumber p298]
[incLude p301]
[Top p302]

incString : String → Function
            — defun incString —

(defun incString (s)
  (declare (special Top))
  (incRenumber (incLude 0 (list s) 0 (list "strings") (list Top))))

Call the garbage collector

Call the garbage collector on various platforms.
            — defun reclaim 0 —

#+abcl
(defun reclaim () "Call the garbage collector" (ext::gc))
defun reroot

The reroot function is used to reset the important variables used by the system. In particular, these variables are sensitive to the AXIOM shell variable. That variable is renamed internally to be \$spadroot. The reroot function will change the system to use a new root directory and will have the same effect as changing the AXIOM shell variable and rerunning the system from scratch. Note that we have changed from the NAG distribution back to the original form. If you need the NAG version you can push :tpd on the *features* variable before compiling this file. A correct call looks like:

```
(in-package "BOOT")
(reroot "/spad/mnt/$SYS")
```

where the $SYS variable is the same one set at build time.

For the example call:

```
(REROOT "/research/test/mnt/ubuntu")
```

the variables are set as:

```
$spadroot = "/research/test/mnt/ubuntu"
$relative-directory-list =
```
11.3. HELPER FUNCTIONS

("/.../src/input/
"/doc/msgs/
"/.../src/algebra/
"/.../src/interp/
"/doc/spadhelp")

$directory-list =
("/research/test/mnt/ubuntu/.../src/input/
"/research/test/mnt/ubuntu/doc/msgs/
"/research/test/mnt/ubuntu/.../src/algebra/
"/research/test/mnt/ubuntu/.../src/interp/
"/research/test/mnt/ubuntu/doc/spadhelp")

$relative-library-directory-list = ("/algebra/")

$library-directory-list = ("/research/test/mnt/ubuntu/algebra/")

-msgDatabaseName| = nil

$current-directory = "/research/test/

---

(defun reroot
 (dir)
 (declare (special $spadroot $directory-list $relative-directory-list
 $library-directory-list $relative-library-directory-list
 $current-directory))
 (setq $spadroot dir)
 (setq $directory-list
   (mapcar #'make-absolute-filename $relative-directory-list))
 (setq $library-directory-list
   (mapcar #'make-absolute-filename $relative-library-directory-list))
 (setq $current-directory $spadroot))

---
defvar $current-directory

— initvars —

(defvar $currentLine "A list of the input line history")

________

defun setCurrentLine

Remember the current line. The cases are:

- If there is no $currentLine set it to the input
- Is the current line a string and the input a string? Make them into a list
- Is $currentLine not a cons cell? Make it one.
- Is the input a string? Cons it on the end of the list.
- Otherwise stick it on the end of the list

[$currentLine p??]

setCurrentLine : String → List(String)

— defun setCurrentLine 0 —

(defun |setCurrentLine| (s)
  (declare (special |$currentLine|))
  (cond
    ((null |$currentLine|) (setq |$currentLine| s))
    ((and (stringp |$currentLine|) (stringp s))
     (setq |$currentLine| (list |$currentLine| s)))
    ((not (consp |$currentLine|)) (setq |$currentLine| (cons |$currentLine| s)))
    ((stringp s) (rplacd (last |$currentLine|) (cons s nil)))
    (t (rplacd (last |$currentLine|) s)))
  |$currentLine|)

________

Show the Axiom prompt

[concat p1197]
[substring p256]
11.3. HELPER FUNCTIONS

mkprompt : Void → String
— defun mkprompt —

(defun mkprompt ()
  "Show the Axiom prompt"
  (declare (special |$inputPromptType| |$IOindex| |$interpreterFrameName|))
  (case |$inputPromptType|
    (|none| "")
    (|plain| "-> ")
    (|step| (concat "(" (princ-to-string |$IOindex|) ") -> "))
    (|frame|
     (concat (princ-to-string |$interpreterFrameName|) " "
                (princ-to-string |$IOindex|) " -> "))
    (t (concat (princ-to-string |$interpreterFrameName|) " ["
         (substring (currenttime) 8 nil) " ["
         (princ-to-string |$IOindex|) " -> "]))))

——

defvar $frameAlist
— initvars —

(defun |$frameAlist| nil)
——

defvar $frameNumber
— initvars —

(defun |$frameNumber| 0)
——
defvar $currentFrameNum

— initvars —

(defvar |$currentFrameNum| 0)

——

defvar $EndServerSession

— initvars —

(defvar |$EndServerSession| nil)

——

defvar $NeedToSignalSessionManager

— initvars —

(defvar |$NeedToSignalSessionManager| nil)

——

defvar $sockBufferLength

— initvars —

(defvar |$sockBufferLength| 9217)

——

READ-LINE in an Axiom server system

[coerceFailure p??]
[top-level p??]
11.3. HELPER FUNCTIONS

[spad-reader p??]
/read-line p??]
/addNewInterpreterFrame p18]
/sockSendInt p??]
/sockSendString p??]
/mkprompt p267]
/sockGetInt p??]
/lassoc p??]
/changeToNamedInterpreterFrame p17]
/sockGetString p??]
/unescapeStringsInForm p287]
/protectedEVAL p271]
/executeQuietCommand p272]
/parseAndInterpret p272]
/serverReadLine is-console (vol9)]
/serverSwitch p??]
/$KillLispSystem p??]
/$NonSmnSession p??]
/$SpadCommand p??]
/$QuietSpadCommand p??]
/$MenuServer p??]
/$sockBufferLength p268]
/$LispCommand p??]
/$EndServerSession p268]
/$EndSession p??]
/$SwitchFrames p??]
/$CreateFrameAnswer p??]
/$currentFrameNum p268]
/$frameNumber p267]
/$frameAlist p267]
/$CreateFrame p??]
/$CallInterp p??]
/$EndOfOutput p??]
/$SessionManager p??]
/$NeedToSignalSessionManager p268]
/$EndServerSession p268]
/$SpadServer p133]
[*eof* p246]
[in-stream p1107]

serverReadLine : Stream → String
    — defun serverReadLine —

(defun |serverReadLine| (stream)
  "used in place of READ-LINE in a Axiom server system."
  (let (in-stream *eof* 1 framename currentframe form stringbuf line action)
(declare (special in-stream *eof* |$SpadServer| |$EndServerSession|
|$NeedToSignalSessionManager| |$SessionManager| |$EndOfOutput|
|$CallInterp| |$CreateFrame| |$frameAlist| |$frameNumber|
|$currentFrameNum| |$CreateFrameAnswer| |$SwitchFrames| |$EndSession|
|$EndServerSession| |$LispCommand| |$sockBufferLength| |$MenuServer|
|$QuietSpadCommand| |$SpadCommand| |$NonSmanSession| |$KillLispSystem|))

(force-output)
(if (or (null |$SpadServer|) (null (is-console stream)))
 (read-line stream)
(progn
 (setq in-stream stream)
 (setq *eof* nil)
 (setq line
 (do ()
 ((null (and (null |$EndServerSession|) (null *eof*))) nil)
 (when |$NeedToSignalSessionManager|
 (sockSendInt |$SessionManager| |$EndOfOutput|))
 (setq |$NeedToSignalSessionManager| nil)
 ; see bookvol8 for the constants that serverSwitch returns
 (setq action (serverSwitch))
 (cond
 ( (= action |$CallInterp|)
 (setq l (read-line stream))
 (setq |$NeedToSignalSessionManager| t)
 (return l))
 ( (= action |$CreateFrame|)
 (setq framename (gentemp "frame"))
 (addNewInterpreterFrame framename)
 (setq |$frameAlist|
 (cons (cons |$frameNumber| framename) |$frameAlist|))
 (setq |$currentFrameNum| |$frameNumber|)
 (sockSendInt |$SessionManager| |$CreateFrameAnswer|)
 (sockSendInt |$SessionManager| |$frameNumber|)
 (setq |$frameNumber| (1+ |$frameNumber|))
 (sockSendString |$SessionManager| (mkprompt)))
 ( (= action |$SwitchFrames|)
 (setq |$currentFrameNum| (sockGetInt |$SessionManager|))
 (setq currentframe (lassoc |$currentFrameNum| |$frameAlist|)
 (changeToNamedInterpreterFrame currentframe))
 ( (= action |$EndSession|)
 (setq |$EndServerSession| t))
 ( (= action |$LispCommand|)
 (setq |$NeedToSignalSessionManager| t)
 (setq stringbuf (make-string |$sockBufferLength|))
 (sockGetString |$MenuServer| stringbuf |$sockBufferLength|)
 (setq form (unescapeStringsInForm (read-from-string stringbuf))
 (protectedEVAL form))
 ( (= action |$QuietSpadCommand|)
 (setq |$NeedToSignalSessionManager| t))
 (executeQuietCommand)))
11.3. HELPER FUNCTIONS

```lisp
((= action |$SpadCommand|)
 (setq |$NeedToSignalSessionManager| t)
 (setq stringbuf (make-string 512))
 ([sockGetString| |$MenuServer| stringbuf 512])
 (catch '|coerceFailure|
   (catch '|top_level|
     (catch 'spad_reader
       ([parseAndInterpret| stringbuf])))
   (princ (mkprompt))
   (finish-output))
 ((= action |$NonSmanSession|) (setq |$SpadServer| nil))
 ((= action |$KillLispSystem|) (bye))
 (t nil)))

(cond
 (line line)
 (t '||))))))

defun protectedEVAL

[resetStackLimits p244]
[sendHTErrorSignal p??]

— defun protectedEVAL —

(defun |protectedEVAL| (x)
 (let (val (error t))
   (unwind-protect
     (progn
       (setq val (eval x))
       (setq error nil))
     (when error
       ([resetStackLimits])
       ([sendHTErrorSignal])))
   (unless error val)))

— initvars —

(defvar |$QuietCommand| nil "If true, produce no top level output")
defun executeQuietCommand

When $QuiteCommand$ is true Spad will not produce any output from a top level command

(defun executeQuietCommand ()
  (let (|$QuietCommand| stringBuf)
    (declare (special |$QuietCommand| |$MenuServer|))
    (setq |$QuietCommand| t)
    (setq stringBuf (make-string 512))
    (|sockGetString| |$MenuServer| stringBuf 512)
    (catch '|coerceFailure|
      (catch '|toplevel|
        (catch 'spad_reader (|parseAndInterpret| stringBuf))))))

---

defun parseAndInterpret

(defun parseAndInterpret (str)
  (let (|$InteractiveMode| $boot $spad |$e|)
    (declare (special |$InteractiveMode| $boot $spad |$e|
                    |$InteractiveFrame|))
    (setq |$InteractiveMode| t)
    (setq $boot nil))
11.3. HELPER FUNCTIONS

(defun parseFromString (s)
  (setq s (next #'ncloopParse (next #'lineoftoks (incString s))))
  (unless (StreamNull s) (pf2Sex (macroExpanded (cadar s)))))

— defun parseFromString —

(defun $interpOnly (nil)
  — initvars —

(defun $minivectorNames (nil)
  — initvars —

(setq $spad t)
(setq |$e| |$InteractiveFrame|)
(|processInteractive| (parseFromString str) nil)))
defvar $domPvar

— initvars —

(defvar $domPvar nil)

defvar $compilingMap

$compilingMap: true when compiling a map, used to detect where to THROW when interpret-only is invoked

— initvars —

(defvar $compilingMap ()

defvar $instantRecord

— initvars —

(setq $instantRecord (make-hash-table :test #'eq))

defun processInteractive

Parser Output --> Interpreter

Top-level dispatcher for the interpreter. It sets local variables and then calls processInterac-tive1 to do most of the work. This function receives the output from the parser.

[initializeTimedNames p??]
[qcar p??]
[processInteractive1 p277]
[reportInstantiations p947]
[clrhash p??]
[writeHistModesAndValues p845]
[updateHist p831]
[$op p??]
11.3. HELPER FUNCTIONS

(defun processInteractive | form posnForm|
  (let (|$op| |$Coerce| |$compErrorMessageStack| |$freeVars|
        |$mapList| |$compilingMap| |$compilingLoop|
        |$interpOnly| $whereCacheList| |$timeGlobalName|
        |$StreamFrame| |$declaredMode| |$localVars|
        |$analyzingMapList| |$lastLineInSEQ|
        |$instantCoerceCount| |$instantCanCoerceCount|
        |$instantMmCondCount| |$fortVar|
        |$minivector| |$minivectorCode| |$minivectorNames|
        |$domPvar| |$inRetract| object)
  (declare (special |$op| |$Coerce| |$compErrorMessageStack|
               |$freeVars| |$mapList| |$compilingMap|
               |$compilingLoop| |$interpOnly| $whereCacheList| |$timeGlobalName|
               |$StreamFrame| |$declaredMode| |$localVars|
               |$analyzingMapList| |$lastLineInSEQ|
               |$instantCoerceCount| |$instantCanCoerceCount|
               |$instantMmCondCount| |$fortVar| |$minivector|
               |$minivectorCode| |$minivectorNames| |$domPvar|
               |$inRetract| object))
CHAPTER 11. HANDLING TERMINAL INPUT

(defun initializeTimedNames
  (|initializeTimedNames| |$interpreterTimedNames| |$interpreterTimedClasses|)
)

(if (consp form) ; compute name of operator
  (setq |$op| (qcar form))
  (setq |$op| form))

(setq |$Coerce| nil)
(setq |$compErrorMessageStack| nil)
(setq |$freeVars| nil) ; list of maps being type analyzed
(setq |$mapList| nil) ; true when compiling a map
(setq |$compilingMap| nil) ; true when compiling a loop body
(setq |$compilingLoop| nil) ; true when in interp only mode
(setq |$whereCacheList| nil) ; maps compiled because of where
(setq |$timeGlobalName| '|$compTimeSum|) ; see incrementTimeSum
(setq |$StreamFrame| nil) ; used in printing streams
(setq |$declaredMode| nil) ; weak type propagation for symbols
(setq |$localVars| nil) ; list of local variables in function
(setq |$analyzingMapList| nil) ; names of maps currently being analyzed
(setq |$lastLineInSEQ| t) ; see evalIF and friends
(setq |$instantCoerceCount| 0)
(setq |$instantCanCoerceCount| 0)
(setq |$instantMmCondCount| 0)
(setq |$defaultFortVar| 'x) ; default FORTRAN variable name
(setq |$fortVar| |$defaultFortVar|) ; variable name for FORTRAN output
(setq |$minivector| nil)
(setq |$minivectorCode| nil)
(setq |$minivectorNames| nil)
(setq |$domPvar| nil)
(setq |$inRetract| nil)
(setq object (|processInteractive1| form posnForm))
(unless |$ProcessInteractiveValue|
  (when |$reportInstantiations|
    (|reportInstantiations|)
    (clrhash |$instantRecord|))
    (|writeHistModesAndValues|)
    (|updateHist|)
    object))


---

defvar $ProcessInteractiveValue
11.3. HELPER FUNCTIONS

— initvars —

(defun initvars
  (defvar $ProcessInteractiveValue nil "If true, no output or record")

------

(defun processInteractive1
  (defun processInteractive1 (form posnForm)
    (let ($e object)
      (declare (special $e $ProcessInteractiveValue $InteractiveFrame))
      (setq $e $InteractiveFrame)
      (recordFrame 'system)
      (startTimingProcess 'analysis)
      (setq object (interpretTopLevel form posnForm))
      (stopTimingProcess 'analysis)
      (startTimingProcess 'print)
      (unless $ProcessInteractiveValue
        (recordAndPrint (objValUnwrap object) (objMode object))
      (recordFrame 'normal)
      (stopTimingProcess 'print))
    object))
defun interpretTopLevel

(defun interpretTopLevel (x posnForm)
  (let (savedTimerStack c)
    (declare (special $timedNameStack))
    (setq savedTimerStack (copy $timedNameStack))
    (setq c (catch '|interpreter| (interpret x posnForm)))
    (do ()
      ((equal savedTimerStack $timedNameStack) nil)
      (stopTimingProcess (peekTimedName)))
    (if (eq c '|tryAgain|)
      (interpretTopLevel x posnForm)
      c)))

defvar $genValue

If the $genValue variable is true then evaluate generated code, otherwise leave code unevaluated. If $genValue is false then we are compiling. This variable is only defined and used locally.

initvars

(defun Type analyzes and evaluates expression x, returns object

(defvar $genValue nil "evaluate generated code if true")

defun Type analyzes and evaluates expression x, returns object
11.3. HELPER FUNCTIONS

---

(defun interpret
  (defun interpret (x rootMode posnForm)
    (let (node modeSet newRootMode argVal val)
      (declare (special $genValue $eval $genValue))
      (setq node (mkAtreeWithSrcPos x posnForm))
      (when rootMode (putTarget node rootMode))
      (setq modeSet (bottomUp node))
      (if (null $eval)
        (progn
          modeSet
          (progn

---

defun Dispatcher for the type analysis routines

This is the dispatcher for the type analysis routines. It type analyzes and evaluates the expression x in the rootMode (if non-nil) which may be $EmptyMode. It returns an object if evaluating, and a modeset otherwise. It creates the attributed tree.

---

(defun interpret1
  (defun interpret1 (x rootMode posnForm)
    (let (node modeSet newRootMode argVal val)
      (declare (special $genValue $eval $genValue))
      (setq node (mkAtreeWithSrcPos x posnForm))
      (when rootMode (putTarget node rootMode))
      (setq modeSet (bottomUp node))
      (if (null $eval)
        (progn
          modeSet
          (progn

---

[mkAtreeWithSrcPos p278]
[putTarget p278]
[bottomUp p278]
[getArgValue p278]
[mkObj p446]
[getValue p278]
[interpret2 p280]
[keyedSystemError p278]
[$genValue p278]
[$eval p278]
(if (null rootMode)
  (setq newRootMode (car modeSet))
  (setq newRootMode rootMode))
(setq argVal (getArgValue node newRootMode))
(cond
  ((and argVal (null (genValue)))
   (mkObj argVal newRootMode))
  ((and argVal (setq val (getValue node)))
   (interpret2 val newRootMode posnForm))
  (t
   (keyedSystemError
    "Interpreter code generation failed for expression %s"
    (list x))))))))

defvar $ThrowAwayMode

— initvars —

(defvar $ThrowAwayMode '|$ThrowAwayMode| "interp constant")

defun interpret2

This is the late interpretCoerce. I removed the call to coerceInteractive, so it only does the JENKS cases.

(defun |interpret2| (object m1 posnForm)
  (declare (ignore posnForm))
  (let (x m op ans)
(declare (special $EmptyMode| $ThrowAwayMode|)
 (cond
  ((equal m1 $ThrowAwayMode|) object)
  (t
   (setq x (|objVal| object))
   (setq m (|objMode| object))
   (cond
    ((equal m $EmptyMode|)
     (cond
      ((and (consp x)
        (progn (setq op (qcar x)) t)
        (|member| op '(map stream)))
       (mkObj x m1))
      ((equal m1 $EmptyMode|)
       (mkObj x m))
      (t
       (|systemErrorHere| "interpret2"))))
    m1
    (if (setq ans (|coerceInteractive| object m1))
     ans
     (|throwKeyedMsgCannotCoerceWithValue| x m m1))
    (t object)))))

defvar $runTestFlag
This is referenced by maPrin to stash output by recordAndPrint to not print type/time — initvars —

(defvar $runTestFlag| nil)

defvar $mkTestFlag
This referenced by READLN to stash input by maPrin to stash output by recordAndPrint to write i/o onto $testStream — initvars —

(defvar $mkTestFlag| nil)
defun Result Output Printing

Prints out the value x which is of type m, and records the changes in environment $e$ into $\text{InteractiveFrame}$ $\text{printAnyIfTrue}$ is documented in setvart.boot. It is controlled with the )se me any command.

[output p??]
[putHist p832]
[mkObjWrap p447]
[printTypeAndTime p284]
[printStorage p283]
[printStatisticsSummary p283]
[mkCompanionPage p??]
[recordAndPrintTest p??]
[$outputMode p??]
[$mkTestOutputType p??]
[$runTestFlag p281]
[$e p247]
[$mkTestFlag p281]
[$HTCompanionWindowID p277]
[$QuietCommand p271]
[$printStatisticsSummaryIfTrue p954]
[$printTypeIfTrue p956]
[$printStorageIfTrue p??]
[$printTimeIfTrue p955]
[$Void p645]
[$algebraOutputStream p967]
[$collectOutput p??]
[$EmptyMode p640]
[$printVoidIfTrue p957]
[$outputMode p??]
[$printAnyIfTrue p939]

--- defun recordAndPrint ---

(defun |recordAndPrint| (x md)
(let (|$outputMode| xp mdp mode)
  (declare (special |$outputMode| |$mkTestOutputType| |$runTestFlag| |$e|
               |$mkTestFlag| |$HTCompanionWindowID| |$QuietCommand|
               |$printStatisticsSummaryIfTrue| |$printTypeIfTrue|
               |$printStorageIfTrue| |$printTimeIfTrue| |$Void|
               |$algebraOutputStream| |$collectOutput| |$EmptyMode|
               |$printVoidIfTrue| |$outputMode| |$printAnyIfTrue|))

(cond
  ((and (equal md '(|Any|)) |$printAnyIfTrue|)
   (setq mdp (car x))
   (setq xp (cdr x)))
  (t
   (setq xp (cdr x))))
11.3. HELPER FUNCTIONS

```lisp
(setq mdp md)
(setq xp x))
(setq |$outputMode| md)
(if (equal md |$EmptyModel|)
  (setq mode (|quadSch|))
  (setq mode md))
(when (or (not (equal md |$Void|)) |$printVoidIfTrue|)
  (unless |$collectOutput| (terpri |$algebraOutputStream|))
  (unless |$QuietCommand| (|output| xp mdp)))
(|putHist| % 'value| (mkObjWrap x md) |$e|)
(when (or |$printTimeIfTrue| |$printTypeIfTrue|)
  (|printTypeAndTime| xp mdp))
(when |$printStorageIfTrue| (|printStorage|))
(when |$printStatisticsSummaryIfTrue| (|printStatisticsSummary|))
(when (integerp |$HTCompanionWindowID|) (|mkCompanionPage| md))
(cond
  (|$mkTestFlag| (|recordAndPrintTest| md))
  (|$runTestFlag|
    (setq |$mkTestOutputType| md)
    '|done|)
  (t '|done|)))
```

defun printStatisticsSummary

```lisp
[sayKeyedMsg p27]
[statisticsSummary p??]
[$collectOutput p??]
```

--- defun printStatisticsSummary ---

```lisp
(defun |printStatisticsSummary| ()
  (declare (special |$collectOutput|))
  (unless |$collectOutput|
    (|sayKeyedMsg| "%rjon Summary: %1 %rjoff" (list (|statisticsSummary|)))))
```

---

defun printStorage

```lisp
[makeLongSpaceString p??]
[$interpreterTimedClasses p??]
[$collectOutput p??]
[$interpreterTimedNames p??]
```
— defun printStorage —

(defun printStorage ()
  (declare (special $interpreterTimedClasses $collectOutput
                 $interpreterTimedNames))
  (unless $collectOutput
    (sayKeyedMsg "%rjon Storage: %1 %rjoff"
      (list (makeLongSpaceString
             $interpreterTimedNames
             $interpreterTimedClasses))))

— defun printTypeAndTime —

(defun printTypeAndTime (x m)
  (let (xp mp timeString result)
    (declare (special $outputLines $collectOutput $printTypeIfTrue
                 $printTimeIfTrue $outputLines
                 $interpreterTimedNames $interpreterTimedClasses))
    (cond ((and (consp m) (eq (qcar m) '|Union|))
           (setq xp (retract (mkObjWrap x m)))
           (setq mp (objMode xp))
           (setq m

11.3. HELPER FUNCTIONS

```lisp
(cons 'Union
  (append (dolist (arg (qcdr m) (nreverse result))
    (when (|sameUnionBranch| arg mp) (push arg result)))
  (list "."))))

(defun printAsTeX (x)
  (declare (special |$texOutputStream|))
  (princ x |$texOutputStream|))
```

defun sameUnionBranch

```lisp
sameUnionBranch(uArg, m) ==
  uArg is [":", , t] => t = m
  uArg = m
```
— defun sameUnionBranch 0 —

(defun |sameUnionBranch| (uArg m)
  (let (t1 t2 t3)
    (cond
      ((and (consp uArg)
            (eq (qcar uArg) '|:|))
       (progn
         (setq t1 (qcdr uArg))
         (and (consp t1)
          (progn
            (setq t2 (qcdr t1))
            (and (consp t2)
             (eq (qcdr t2) nil)
              (progn (setq t3 (qcar t2)) t))))))
      (equal t3 m))
      (t (equal uArg m))))

——

defun msgText

[segmentKeyedMsg p28]
[substituteSegmentedMsg p??]
[flowSegmentedMsg p??]
[$linelength p983]
[$margin p983]

— defun msgText —

(defun |msgText| (key args)
  (let (msg)
    (declare (special $linelength $margin))
    (setq msg (|segmentKeyedMsg| key))
    (setq msg (|substituteSegmentedMsg| msg args))
    (setq msg (|flowSegmentedMsg| msg $linelength $margin))
    (apply #'concat (mapcar #'princ-to-string (cdar msg))))

——

defun Right-justify the Type output

[fillerSpaces p242]
[$linelength p983]
— defun justifyMyType —
(defun justifyMyType (arg)
  (let (len)
    (declare (special $linelength))
    (setq len (len arg))
    (if (> len $linelength)
      arg
      (concat (fillerSpaces (- $linelength len)) arg))))

— defun Destructively fix quotes in strings —
(defun unescapeStringsInForm (form)
  (let (str)
    (declare (special $funnyBacks $funnyQuote))
    (cond
      ((stringp form)
       (setq str (nsubstitute #" $funnyQuote form))
       (nsubstitute #\ $funnyBacks str))
      ((consp form)
       (unescapeStringsInForm (car form))
       (unescapeStringsInForm (cdr form))
       form)
      (t form))))

— defun intloopInclude —
(defun intloopInclude (name n)
  "Include a file into the stream"
  (with-open-file (st name) (intloopInclude0 st name n)))
### defun intloopInclude0

```lisp
(defun intloopInclude0
  (incStream p298
  intloopProcess p289
  next p262
  intloopEchoParse p294
  insertpile p557
  lineoftoks p335
  $lines p??)

---

---

### defun intloopProcess

An example call looks like:

```
3> (intloopProcess 1 T
  (nonnullstream #0=next1 ncloopParse
  (nonnullstream #0# lineoftoks
  (nonnullstream incZip1 incRenumberLine
  (nonnullstream incIgen1 0 ("1") 0 ("strings") (1))
  (nonnullstream incIgen1 (0))))))
```

which was constructed intloopProcessString(p261). This call says we are processing the first input, in this case “1”. It is interactive. The third argument, the delay, contains the information to drive the rest of the process. [StreamNull p555]

[pfAbSynOp? p634]
[setCurrentLine p266]
[tokPart p635]
[intloopProcess p289]
[intloopSpadProcess p289]
11.3. HELPER FUNCTIONS

intloopProcess : (StepNo,Boolean,Delay) → StepNo
— defun intloopProcess —

(defun intloopProcess (stepno interactive delay)
  (let (ptree lines t1)
    (declare (special $systemCommandFunction!))
    (cond
      ((StreamNull delay) stepno)
      (t
        (setq t1 (car delay))
        (setq lines (car t1))
        (setq ptree (cadr t1))
        (cond
          ((pfAbSymOp? ptree 'command))
            (when interactive (setCurrentLine (tokPart ptree)))
            (funcall $systemCommandFunction! (tokPart ptree))
            (intloopProcess stepno interactive (cdr delay)))
      (t
        (intloopProcess
          (intloopSpadProcess stepno lines ptree interactive)
          interactive (cdr delay)))))))

defun intloopSpadProcess

[flung p?]
[SpadCompileItem p?]
[intCoerceFailure p255]
[intSpadReader p255]
[ncPutQ p638]
[CatchAsCan p?]
[Catch p?]
[intloopSpadProcess,interp p290]
[currentCarrier p?]
[ncMsgList p250]
[intCoerceFailure p255]
[intSpadReader p255]
[prevCarrier p?]
[stepNo p?]
[NeedToSignalSessionManager p268]
[flung p?]
CHAPTER 11. HANDLING TERMINAL INPUT

---

```lisp
(defun intloopSpadProcess (stepNo lines ptree interactive?)
  (let ((|$stepNo| result cc)
        (declare (special |$stepNo| |$prevCarrier| |$intSpadReader| |flung|
                    |$intCoerceFailure| |$ncMsgList| |$currentCarrier|
                    |$NeedToSignalSessionManager|))
    (setq |$stepNo| stepNo)
    (setq |$currentCarrier| (setq cc (list '|carrier|)))
    (|ncPutQ| cc '|stepNumber| stepNo)
    (|ncPutQ| cc '|messages| |$ncMsgList|)
    (|ncPutQ| cc '|lines| lines)
    (setq |$ncMsgList| nil)
    (setq result
      (catch '|SpadCompileItem|
        (catch |$intCoerceFailure|
          (catch |$intSpadReader|
            (intloopSpadProcess,interp cc ptree interactive?)))))
    (setq |$NeedToSignalSessionManager| t)
    (setq |$prevCarrier| |$currentCarrier|)
    (cond
      ((eq result '|ncEnd|) stepNo)
      ((eq result '|ncError|) stepNo)
      ((eq result '|ncEndItem|) stepNo)
      (t (1+ stepNo)))))

---

defun intloopSpadProcess,interp

[ncConversationPhase p292]
[ncEltQ p638]
[ncError p293]

---

(defun intloopSpadProcess,interp (cc ptree interactive?)
  (|ncConversationPhase| #'|phParse| (list cc ptree))
  (|ncConversationPhase| #'|phMacro| (list cc))
  (|ncConversationPhase| #'|phIntReportMsgs| (list cc interactive?))
  (|ncConversationPhase| #'|phInterpret| (list cc))
  (unless (eql (length (|ncEltQ| cc '|messages|)) 0) (|ncError|)))
11.3. HELPER FUNCTIONS

defun phParse

TPDHERE: The pform function has a leading percent sign

phParse: carrier[tokens,...] -> carrier[ptree, tokens,...]

[ncPutQ p638]

— defun phParse —

(defun |phParse| (carrier ptree)
  (|ncPutQ| carrier '|ptree| ptree)
  'ok)

defun phIntReportMsgs

carrier[lines,messages,..]-> carrier[lines,messages,..]

[ncEltQ p638]
[ncPutQ p638]
[processMsgList p591]
[$erMsgToss p??]

— defun phIntReportMsgs —

(defun |phIntReportMsgs| (carrier interactive?)
  (declare (ignore interactive?))
  (let (nerr msgs lines)
    (declare (special |$erMsgToss|))
    (cond
      ($|erMsgToss| 'ok)
      (t
       (setq lines (|ncEltQ| carrier '|lines|))
       (setq msgs (|ncEltQ| carrier '|messages|))
       (setq nerr (length msgs))
       (|ncPutQ| carrier '|ok?| (eql nerr 0))
       (cond
        ((eql nerr 0) 'ok)
        (t
         (|processMsgList| msgs lines)
         (|sayKeyedMsg| "%1 error(s) parsing " (list nerr))
         'ok))))))

———
defun phInterpret

(ncEltQ p638)
(intInterpretPform p292)
(ncPutQ p638)

— defun phInterpret —

(defun phInterpret (carrier)
  (let (val ptree)
    (setq ptree (ncEltQ carrier 'ptree))
    (setq val (intInterpretPform ptree))
    (ncPutQ carrier 'value val)))

—

defun intInterpretPform

(processInteractive p274)
(zeroOneTran p292)
(pf2Sex p529)

— defun intInterpretPform —

(defun intInterpretPform (pf)
  (processInteractive (zeroOneTran (pf2Sex pf)) pf))

—

defun zeroOneTran

(nsubst p??)

— defun zeroOneTran 0 —

(defun zeroOneTran (sex)
  (nsubst '$EmptyMode ' sex))

—

defun ncConversationPhase

(ncConversationPhase,wrapup p293)
($ncMsgList p250)
— defun ncConversationPhase —

(defun |ncConversationPhase| (fn args)
  (let (($ncMsgList| carrier)
         (declare (special $ncMsgList|))))
  (setq carrier (car args))
  (setq $ncMsgList| nil)
  (unwind-protect
      (apply fn args)
      (|ncConversationPhase,wrapup| carrier))))

— defun ncConversationPhase,wrapup —

(defun |ncConversationPhase,wrapup| (carrier)
  (declare (special $ncMsgList|))
  ((lambda (Var5 m)
      (loop
        (cond
          ((or (atom Var5) (progn (setq m (car Var5)) nil))
           (return nil))
          (t
           (|ncPutQ| carrier '|messages| (cons m (|ncEltQ| carrier '|messages|))))
           (setq Var5 (cdr Var5))))
           $ncMsgList| nil))

— defun ncError —

(defun |ncError| ()
  (throw '|SpadCompileItem| '|ncError|))
defun intloopEchoParse

(ncloopDQlines p296)
(setCurrentLine p266)
(mkLineList p295)
(ncloopPrintLines p294)
(upParse p365)
(dqToList p566)
($EchoLines p??)
($lines p??)

— defun intloopEchoParse —

(defun intloopEchoParse (s)
  (let ((cudr lines stream dq t1))
    (declare (special $EchoLines $lines))
    (setq t1 (car s))
    (setq dq (car t1))
    (setq stream (cadr t1))
    (setq t1 (ncloopDQlines dq $lines))
    (setq lines (car t1))
    (setq cudr (cadr t1))
    (setCurrentLine (mkLineList lines))
    (when $EchoLines (ncloopPrintLines lines))
    (setq $lines cudr)
    (cons (list (list lines (npParse (dqToList dq)))) (cdr s))))

— defun ncloopPrintLines —

(defun ncloopPrintLines (lines)
  ((lambda (Var4 line)
     (loop
       (cond
         ((or (atom Var4) (progn (setq line (car Var4)) nil))
          (return nil))
         (t (write-line (cdr line))))
       (setq Var4 (cdr Var4)))))

— defun ncloopPrintLines 0 —

(defun ncloopPrintLines (lines)
  ((lambda (Var4 line)
     (loop
       (cond
         ((or (atom Var4) (progn (setq line (car Var4)) nil))
          (return nil))
         (t (write-line (cdr line))))
       (setq Var4 (cdr Var4))))

;ncloopPrintLines lines ==
; for line in lines repeat WRITE_-LINE CDR line
; WRITE_-LINE " "

— defun ncloopPrintLines 0 —
11.3. HELPER FUNCTIONS

lines nil)
(write-line " ")

---

defun mkLineList

; mkLineList lines ==
;  l := [CDR line for line in lines | nonBlank CDR line]
;  #l = 1 => CAR l
;  l

(defun mkLineList |lines|)
(let (l)
(setq l
((lambda (Var2 Var1 line)
  (loop
    (cond
      ((or (atom Var1) (progn (setq line (car Var1)) nil))
        (return (nreverse Var2)))
      (t
        (and (|nonBlank| (cdr line))
          (setq Var2 (cons (cdr line) Var2)))))))
  (setq Var1 (cdr Var1))))
nil lines nil))
(cond
((eql (length l) 1) (car l))
(t l)))

---

defun nonBlank

; nonBlank str ==
; value := false
; for i in 0..MAXINDEX str repeat
;  str.i ^= char " " =>
;  value := true
;  return value
; value

(defun nonBlank 0)
(defun \nonBlank\ (str)
  (let (value)
    ((lambda (Var3 i)
       (loop
        (cond
          ((> i Var3) (return nil))
          (t
           (cond
            ((not (equal (elt str i) #\Space))
             (identity (progn (setq value t) (return value)))))))
        (setq i (+ i 1))))
      (maxindex str) 0)
    value))

defun ncloopDQlines

[StreamNull p555]
[poGlobalLinePosn p296]
[tokPosn p635]
[streamChop p297]

— defun ncloopDQlines —

(defun \ncloopDQlines\ (dq stream)
  (let (b a)
    (\StreamNull\ stream)
    (setq a (\poGlobalLinePosn\ (\tokPosn\ (cadr dq)))
      b (\poGlobalLinePosn\ (caar stream))
    (\streamChop\ (+ (- a b) 1) stream))

—

defun poGlobalLinePosn

[lnGlobalNum p568]
[poGetLineObject p583]
[ncBug p590]

— defun poGlobalLinePosn —

(defun \poGlobalLinePosn\ (posn)
  (if posn
    (\lnGlobalNum\ (\poGetLineObject\ posn)))
defun streamChop

Note that changing the name “lyne” to “line” will break the system. I do not know why. The symptom shows up when there is a file with a large contiguous comment spanning enough lines to overflow the stack.

(defun |streamChop| |n| |s|
  (let (|d| |c| |lyne| |b| |tmp1|)
    (cond
      ((StreamNull |s|) (list nil nil))
      (eql |n| 0) (list nil |s|))
      (t
        (setq |tmp1| (streamChop (- |n| 1) (cdr |s|)))
        (setq |a| (car |tmp1|))
        (setq |b| (cadr |tmp1|))
        (setq |lyne| (car |s|))
        (setq |c| (ncloopPrefix? "command" (cdr |lyne|)))
        (setq |d| (cons (car |lyne|) (cond (c |c|) (t (cdr |lyne|))))))
      (list (cons |d| |a|) |b|))))

— defun streamChop —

defun ncloopInclude0

(defun |ncloopInclude0| |st| |name| |n|
  (...))

— defun ncloopInclude0 —

(defun |ncloopInclude0| (st name n)
(let ([lines])
  (declare (special lines))
  (setq lines (incStream st name))
  (ncloopProcess n nil
    (next #'ncloopEchoParse
      (next #'insertpile
        (next #'lineoftoks
          lines)))))

---

defun incStream

[incRenumber p298]
[incLude p301]
[incRgen p329]
[Top p302]

---

defun incRenumber

[incZip p299]
[inclgen p299]

incRenumber : Delay → Delay

---

defun incZip

Axiom “zips” a function together with two delays into a delay.
11.3. HELPER FUNCTIONS

incZip : (Function, Delay, Delay) → Delay
— defun incZip —

(defun incZip (function delay1 delay2)
  (Delay #'incZip1 (list function delay1 delay2)))

defun incZip1

incZip1 : Delay → ParsePair
— defun incZip1 —

(defun incZip1 (&rest delayArg)
  (let (function delay1 delay2)
    (setq function (car delayArg))
    (setq delay1 (cadr delayArg))
    (setq delay2 (caddr delayArg))
    (cond
      ((StreamNull delay1) StreamNil)
      ((StreamNull delay2) StreamNil)
      (t
        (cons
          (funcall function (car delay1) (car delay2))
          (incZip1 function (cdr delay1) (cdr delay2)))))))

defun incIgen

incIgen : Integer → Delay
— defun incIgen —

(defun incIgen (int)
  (Delay #'incIgen1 (list int)))
defun incIgen1

(defun incIgen1 (x &rest z)
  (let (n)
    (setq n (car x))
    (setq n (+ n 1))
    (cons n (incIgen1 n))))

defun incRenumberLine

(defun incRenumberLine (xl gno)
  (let (l)
    (setq l (incRenumberItem (elt xl 0) gno))
    (incHandleMessage xl)
    l))

defun incRenumberItem

(defun incRenumberItem (f i)
  (let (l)
    (setq l (caar f))
    (incSetGlobalNum l i) f))
11.3. HELPER FUNCTIONS

defun incHandleMessage

(ncSoftError p574)
(ncBug p590)

— defun incHandleMessage 0 —

(defun |incHandleMessage| (x)
"Message handling for the source includer"
(let ((msgtype (elt (elt x 1) 1))
  (pos (car (elt x 0)))
  (key (car (elt (elt x 1) 0)))
  (args (cadr (elt (elt x 1) 0))))

(cond
  ((eq msgtype '|none|) 0)
  ((eq msgtype '|error|) (|ncSoftError| pos key args))
  ((eq msgtype '|warning|) (|ncSoftError| pos key args))
  ((eq msgtype '|say|) (|ncSoftError| pos key args))
  (t (|ncBug| key args))))

defun incLude

This function takes

1. eb – in Integer
2. ss – a list of strings
3. ln – an Integer
4. ufos – a list of strings
5. states – a list of integers

and constructs a call to Delay(p329).

[Delay p329]
[incLude1 p306]

incLude : (Int,List(String),Int,List(String),List(Int)) → Delay
— defun incLude —

(defun |incLude| (eb ss ln ufos states)
  (|Delay| #'|incLude1| (list eb ss ln ufos states)))

———
defmacro Rest

—— defmacro Rest ——

(defmacro |Rest| ()
"used in include1 for parsing; s is not used."
'(|include| eb (cdr ss) lno ufos states))

——

defvar Top

—— initvars ——

(defvar |Top| 1 "used in include1 for parsing")

——

defvar IfSkipToEnd

—— initvars ——

(defvar |IfSkipToEnd| 10 "used in include1 for parsing")

——

defvar IfKeepPart

—— initvars ——

(defvar |IfKeepPart| 11 "used in include1 for parsing")

——

defvar IfSkipPart

—— initvars ——
11.3. HELPER FUNCTIONS

(defvar |IfSkipPart| 12 "used in include1 for parsing")

---

defvar ElseIfSkipToEnd

— initvars —

(defvar |ElseSkipToEnd| 30 "used in include1 for parsing")

---

defvar ElseIfKeepPart

— initvars —

(defvar |ElseKeepPart| 21 "used in include1 for parsing")

---

defvar ElseIfSkipPart

— initvars —

(defvar |ElseSkipPart| 22 "used in include1 for parsing")

---

defvar ElseSkipToEnd

— initvars —

(defvar |ElseSkipToEnd| 30 "used in include1 for parsing")

---
defvar ElseKeepPart

— initvars —

(defvar |ElseKeepPart| 31 "used in include1 for parsing")

defun Top?
[quotient p??]

— defun Top? 0 —

(defun |Top?| (|st|)
"used in include1 for parsing"
(eql (quotient |st| 10) 0))

defun If?
[quotient p??]

— defun If? —

(defun |If?| (|st|)
"used in include1 for parsing"
(eql (quotient |st| 10) 1))

defun Elseif?
[quotient p??]

— defun Elseif? —

(defun |Elseif?| (|st|)
"used in include1 for parsing"
(eql (quotient |st| 10) 2))
defun Else?
[quotient p??]

— defun Else? —

(defun |Else?| (|st|)
  "used in include1 for parsing"
  (eql (quotient |st| 10) 3))

———

defun SkipEnd?
[remainder p??]

— defun SkipEnd? —

(defun |SkipEnd?| (|st|)
  "used in include1 for parsing"
  (eql (remainder |st| 10) 0))

———

defun KeepPart?
[remainder p??]

— defun KeepPart? —

(defun |KeepPart?| (|st|)
  "used in include1 for parsing"
  (eql (remainder |st| 10) 1))

———

defun SkipPart?
[remainder p??]

— defun SkipPart? —
(defun |SkipPart?| (|st|)
  "used in incLude1 for parsing"
  (eql (remainder |st| 10) 2))

----------

defun Skipping?

[KeepPart? p305]

— defun Skipping? —

(defun |Skipping?| (|st|)
  "used in incLude1 for parsing"
  (null (|KeepPart?| |st|)))

----------

defun incLude1

[StreamNull p555]
[Top? p304]
[xlPrematureEOF p310]
[Skipping? p306]
[xlSkip p314]
[Rest p302]
[xlOK p311]
[xlOK1 p311]
[concat p1197]
[incCommandTail p327]
[xlSay p314]
[xlNoSuchFile p315]
[xlCannotRead p316]
[incActive? p328]
[xlFileCycle p317]
[incLude p301]
[incFileInput p327]
[incAppend p312]
[inclFname p327]
[xlConActive p318]
[xlConStill p319]
[incConsoleInput p328]
[incNConsoles p328]
11.3. HELPER FUNCTIONS

---

(defun incLude1 (&rest z)
  (let (pred s1 n tail head includee fn1 info str state lno states
        ufos ln ss eb)
    (setq eb (car z))
    (setq ss (cadr . (z)))
    (setq ln (caddr . (z)))
    (setq ufos (cadddr . (z)))
    (setq states (car (cddddr . (z))))
    (setq lno (+ ln 1))
    (setq state (elt states 0))
    (cond
      (((|StreamNull| ss) ss)
       (cond
         (null (|Top?| state))
         (cons (|xlPrematureEOF| eb "")--premature end" lno ufos)
           (|StreamNil|))
         (t (|StreamNil|)))
      (t (progn
          (setq str (expand-tabs (car ss)))
          (setq info (|incClassify| str))
          (cond
            ((null (elt info 0))
             (cond
              (((|Skipping?| state)
                (cons (|xlSkip| eb str lno (elt ufos 0)) (|Rest|)))
              (t (cons (|xlOK| eb str lno (elt ufos 0)) (|Rest|)))))
            (t ((equal (elt info 2) "other")
                (cond
                )))
          )))
        )))
((|Skipping?| state)
  (cons (|xlSkip| eb str lno (elt ufos 0)) (|Rest|)))
(t
  (cons
   (|xlOK1| eb str (concat ")command" str) lno (elt ufos 0))
   (|Rest|))))
((equal (elt info 2) "say")
  (cond
   ((|Skipping?| state)
    (cons (|xlSkip| eb str lno (elt ufos 0)) (|Rest|)))
   (t
    (progn
     (setq str (|incCommandTail| str info))
     (cons (|xlSay| eb str lno ufos str)
       (cons (|xlOK| eb str lno ufos str)
         (cons (|Rest|))))))))
((equal (elt info 2) "include")
  (cond
   ((|Skipping?| state)
    (cons (|xlSkip| eb str lno (elt ufos 0)) (|Rest|)))
   (t
    (progn
     (setq fn1 (|inclFname| str info))
     (cond
      (null fn1)
      (cons (|xlNoSuchFile| eb str lno ufos fn1) (|Rest|))
      (null (probe-file fn1))
      (cons (|xlCannotRead| eb str lno ufos fn1) (|Rest|))
      (|incActive?| fn1 ufos)
      (cons (|xlFileCycle| eb str lno ufos fn1) (|Rest|)))
      (t
       (progn
        (setq includee
          (|inclLude| (+ eb (elt info 1))
          (|incFileInput| fn1 ufos)
          0
          (cons fn1 ufos)
          (cons [Top] states)))
        (cons (|xlOK| eb str lno (elt ufos 0))
          (|incAppend| includee (|Rest|))))))))
((equal (elt info 2) "console")
  (cond
   ((|Skipping?| state)
    (cons (|xlSkip| eb str lno (elt ufos 0)) (|Rest|)))
   (t
    (progn
     (setq head
       (|inclLude| (+ eb (elt info 1))
       (|incConsoleInput|)
       0
       (cons "console" ufos))
11.3. HELPER FUNCTIONS

(cons |Top| states)))
=setq tail (|Rest|)
=setq n (|incNConsoles| ufos))
(cond
((< 0 n)
 (setq head
 (cons (|xlConActive| eb str lno ufos n) head))
(setq tail
 (cons (|xlConStill| eb str lno ufos n) tail))))
(setq head (cons (|xlConsole| eb str lno ufos) head))
(cons (|xlOK| eb str lno (elt ufos 0))
 (|incAppend| head tail))))

((equal (elt info 2) "fin")
 (cond
 (|Skipping?| state)
 (cons (|xlSkippingFin| eb str lno ufos) (|Rest|)))
 (null (|Top?| state))
 (cons (|xlPrematureFin| eb str lno ufos) |StreamNil|))
 (t
 (cons (|xlOK| eb str lno (elt ufos 0)) |StreamNil|))))

((equal (elt info 2) "assert")
 (cond
 (|Skipping?| state)
 (cons (|xlSkippingFin| eb str lno ufos) (|Rest|)))
 (t
 (progn
 (|assertCond| str info)
 (cons (|xlOK| eb str lno (elt ufos 0))
 (|incAppend| includee (|Rest|)))))

((equal (elt info 2) "if")
 (progn
 (setq s1
 (cond
 (|Skipping?| state) |IfSkipToEnd|)
 (t
 (cond
 (|ifCond| str info) |IfKeepPart|)
 (t |IfSkipPart|)))
 (cons (|xlOK| eb str lno (elt ufos 0))
 (|incLude| eb (cdr ss) lno ufos (cons s1 states))))

((equal (elt info 2) "elseif")
 (cond
 (and (null (|If?| state)) (null (|Elseif?| state)))
 (cons (|xlIfSyntax| eb str lno ufos info states)
 |StreamNil|))
 (t
 (cond
 (|SkipEnd?| state)
 (|KeepPart?| state)
 (|SkipPart?| state))
(setq s1
 (cond
  (((&|SkipPart?| state)
     (setq pred (&|ifCond| str info))
     (cond
      (pred &|ElseIfKeepPart|)
      (t &|ElseIfSkipPart|)))
     (t &|ElseIfSkipToEnd|)))
  (cons (|xlOK| eb str lno (elt ufos 0))
     (|incLude| eb (cdr ss) lno ufos (cons s1 (cdr states))))))
 (t
  (cons (|xlIfBug| eb str lno ufos) |StreamNil|)))))))
((equal (elt info 2) "else")
 (cond
  ((and (null (&|If?| state)) (null (&|Elseif?| state)))
  (cons (|xlIfSyntax| eb str lno ufos info states)
     |StreamNil|))
  (t
  (cond
   ((or (&|SkipEnd?| state)
     (&|KeepPart?| state)
     (&|SkipPart?| state))
     (setq s1
        (cond ((&|SkipPart?| state) &|ElseKeepPart|) (t &|ElseSkipToEnd|)))
        (cons (|xlOK| eb str lno (elt ufos 0))
          (|incLude| eb (cdr ss) lno ufos (cons s1 (cdr states))))))
     (t
      (cons (|xlIfBug| eb str lno ufos) |StreamNil|))))))
  (equal (elt info 2) "endif")
  (cond
   ((&|Top?| state)
     (cons (|xlIfSyntax| eb str lno ufos info states)
        |StreamNil|))
   (t
    (cons (|xlOK| eb str lno (elt ufos 0))
      (|incLude| eb (cdr ss) lno ufos (cdr states))))
    (t (cons (|xlCmdBug| eb str lno ufos) |StreamNil|))))))))

defun xlPrematureEOF
[xlMsg p311][inclmsgPrematureEOF p313]
— defun xlPrematureEOF —
(defun |xlPrematureEOF| (eb str lno ufos)
11.3. HELPER FUNCTIONS

(|xlMsg| eb str lno (elt ufos 0)
 (list (|inclmsgPrematureEOF| (elt ufos 0)) '|error|)))

---

defun xlMsg
[incLine p312]

— defun xlMsg —

(defun |xlMsg| (extrablanks string localnum fileobj mess)
 (let ((globalnum -1))
  (list (incLine extrablanks string globalnum localnum fileobj) mess)))

---

defun xlOK
[xlOK1 p311]

— defun xlOK —

(defun |xlOK| (extrablanks string localnum fileobj)
  (|xlOK1| extrablanks string string localnum fileobj))

---

defun xlOK1
[incLine1 p313]

— defun xlOK1 —

(defun |xlOK1| (extrablanks string string1 localnum fileobj)
 (let ((globalnum -1))
  (list (incLine1 extrablanks string string1 globalnum localnum fileobj)
        (list nil '|none|))))
defun incAppend

[Delay p329]
[incAppend1 p312]

— defun incAppend —

(defun |incAppend| (x y)
  ((|Delay| #'|incAppend1| (list x y)))

defun incAppend1

[StreamNull p555]
[incAppend p312]

— defun incAppend1 —

(defun |incAppend1| (&rest z)
  (let (y x)
    (setq x (car z))
    (setq y (cadr z))
    (cond
      ((|StreamNull| x)
       (cond ((|StreamNull| y) |StreamNil|) (t y)))
      (t
       (cons (car x) (|incAppend| (cdr x) y))))))

defun incLine

[incLine1 p313]

— defun incLine —

(defun incLine (extrablanks string globalnum localnum fileobj)
  (incLine1 extrablanks string string globalnum localnum fileobj))
defun incLine1

(defun incLine1 (extrablanks string string1 globalnum localnum fileobj)
  (cons
   (cons (|lnCreate| extrablanks string globalnum localnum fileobj) 1) string1))

defun inclmsgPrematureEOF

(defun inclmsgPrematureEOF (ufo)
  (list
   (format nil
    "File %1f ended where at least one )endif was still needed.
    An appropriate number of )endif lines has been assumed."
    (list (|theorigin| ufo)))))

defun theorigin

(defun theorigin (x) (list #'|porigin| x))

defun porigin

(defun porigin (pfname)
  — defun porigin —
  (list #'|porigin| x))
(defun porigin (x)
  (if (stringp x)
      x
      (pname x)))

—

defun ifCond

[MakeSymbol p?]
[incCommandTail p 327]
[$inclAssertions p??]

  — defun ifCond —

(defun ifCond (s info)
  (let (word)
    (declare (special "$inclAssertions$"))
    (setq word
      (MakeSymbol (string-trim *whitespace* (incCommandTail s info)))
    (member word "$inclAssertions$")))

—

defun xlSkip

[incLine p 312]
[concat p 1197]

  — defun xlSkip —

(defun xlSkip (extrablanks str localnum fileobj)
  (let ((string (concat "-- Omitting:" str) (globalnum -1))
    (list
      (incLine extrablanks string globalnum localnum fileobj)
    (list nil ´'none)))))

—

defun xlSay

[xlMsg p 311]
[inclmsgSay p 315]
11.3. HELPER FUNCTIONS

---

**defun xlSay**

(defun xlSay (eb str lno ufos x)
  (xlMsg eb str lno (elt ufos 0) (list (inclmsgSay x) 'say)))

---

**defun inclmsgSay**

[theid p315]

---

(defun inclmsgSay (str)
  (list "%1f" (list (theid str)))))

---

**defun theid**

---

(defun theid 0 (a) (list #'identity a))

---

**defun xlNoSuchFile**

[xlMsg p311]
[inclmsgNoSuchFile p316]

---

(defun xlNoSuchFile (eb str lno ufos fn)
  (xlMsg eb str lno (elt ufos 0) (list (inclmsgNoSuchFile fn) 'error)))

---
defun inclmsgNoSuchFile

[thefname p316]

— defun inclmsgNoSuchFile —

(defun inclmsgNoSuchFile (fn)
  (list "The )include file %1f does not exist." (list (thefname fn))))

——

defun thefname

[pfname p316]

— defun thefname 0 —

(defun thefname (x) (list #'pfname x))

——

defun pfname

[PathnameString p??]

— defun pfname —

(defun pfname (x) (list'PathnameString x))

——

defun xlCannotRead

[xlMsg p311]
[inclmsgCannotRead p317]

— defun xlCannotRead —

(defun xlCannotRead (eb str lno ufos fn) 
  (list (xlMsg eb str lno (elt ufos 0) (list (inclmsgCannotRead fn) 'error))))

——
11.3. HELPER FUNCTIONS

defun inclmsgCannotRead

[thefname p316]

    — defun inclmsgCannotRead —

(defun |inclmsgCannotRead| (fn)
  (list "The \include file %1f exists, but cannot be read."
    (list (|thefname| fn))))

---------

defun xlFileCycle

[xlMsg p311]
[inclmsgFileCycle p317]

    — defun xlFileCycle —

(defun |xlFileCycle| (eb str lno ufos fn)
  (|xlMsg| eb str lno (elt ufos 0)
    (list (|inclmsgFileCycle| ufos fn) '|error|)))

---------

defun inclmsgFileCycle

;inclmsgFileCycle(ufos,fn) ==
;   flist := [porigin n for n in reverse ufos]
;   f1    := porigin fn
;   cycle := [:[:[n,""==">"] for n in flist], f1]
;   ['SCI0004, [%id cycle, %id f1] ]

[porigin p313]
[theid p315]

    — defun inclmsgFileCycle —

(defun |inclmsgFileCycle| (ufos fn)
  (let (cycle f1 flist)
    (setq flist
      (((lambda (Var8 Var7 n)
        (loop
(cond
  ((or (atom Var7) (progn (setq n (car Var7)) nil))
   (return (nreverse Var8)))
  (t
   (setq Var8 (cons (|porigin| n) Var8)))
  (setq Var7 (cdr Var7)))
nil (reverse ufos) nil))
(setq f1 (|porigin| fn))
(setq cycle
  (append
   ((lambda (Var10 Var9 n)
      (loop
       (cond
        ((or (atom Var9) (progn (setq n (car Var9)) nil))
         (return (nreverse Var10)))
        (t
         (setq Var10 (append (reverse (list n "==>")) Var10))))
        (setq Var9 (cdr Var9)))
nil flist nil)
  (cons f1 nil)))
(list
  (format nil
    "There is a cycle in the )include files: %i %l %1f %u %l. ~
The inner occurrence of %2f has not been included.")
  (list (|theid| cycle) (|theid| f1))))

----------

defun xlConActive

[xlMsg p311]
[inclmsgConActive p318]

— defun xlConActive —

(defun |xlConActive| (eb str lno ufos n)
  (|xlMsg| eb str lno (elt ufos 0) (list (|inclmsgConActive| n) '|warning|)))

----------

defun inclmsgConActive

[theid p315]

— defun inclmsgConActive —
(defun inclmsgConActive (n)
  (list
   (format nil
    "%1f other )console commands are currently active. ~
    While this new )console command is reading input the others ~
    will have to wait. !
    Remember, each )console command will need a separate )fin."
   )
  (list (theid n))))

defun xlConStill
[xlMsg p311]
[inclmsgConStill p319]

—— defun xlConStill ——

(defun xlConStill| (eb str lno ufos n)
  (xlMsg eb str lno (elt ufos 0) (list (inclmsgConStill n) ’say)))

——

defun inclmsgConStill
[theid p315]

—— defun inclmsgConStill ——

(defun inclmsgConStill| (n)
  (list
   (format nil
    "The current )console command has finished reading. ~
    %1f are still active. Remember, each will need a separate )fin."
   )
  (list (theid n))))

——

defun xlConsole
[xlMsg p311]
[inclmsgConsole p320]

—— defun xlConsole ——
(defun xlConsole (eb str lno ufos)
  (xlMsg eb str lno (elt ufos 0) (list (inclmsgConsole) 'say)))

defun inclmsgConsole

  — defun inclmsgConsole 0 —

  (defun inclmsgConsole ()
    (list "Including source lines from console. Type )fin when done." nil))

— defun xlSkippingFin —

(defun xlSkippingFin (eb str lno ufos)
  (xlMsg eb str lno (elt ufos 0)
    (list (inclmsgFinSkipped) 'warning)))

— defun inclmsgFinSkipped —

(defun inclmsgFinSkipped ()
  (list
    (format nil
      "A )fin command was skipped ~
      (along with everything else) in a false branch of an )if...ENDIF."
      nil)))

— defun inclmsgFinSkipped 0 —
### 11.3. HELPER FUNCTIONS

```lisp
(defun xlPrematureFin

[xlMsg p311]
[inclmsgPrematureFin p321]

--- defun xlPrematureFin ---

(defun xlPrematureFin (eb str lno ufos)
  (xlMsg eb str lno (elt ufos 0)
    (list (inclmsgPrematureFin (elt ufos 0)) 'error)))

---

(defun inclmsgPrematureFin

[theorigin p313]

--- defun inclmsgPrematureFin ---

(defun inclmsgPrematureFin (ufo)
  (list
    (format nil
      "A )fin command has been given in %1f where at least one )endif ~
       was still needed. ~
       An appropriate number of )endif lines have been assumed."
    )
    (list (theorigin ufo))))

---

(defun assertCond

[MakeSymbol p??]
[incCommandTail p327]
[$inclAssertions p??]
[*whitespace* p246]

--- defun assertCond ---

(defun assertCond (s info)
  (let (word)
    (declare (special $inclAssertions whitespace))
    (setq word
      (MakeSymbol (string-trim whitespace
       (incCommandTail s info)))))
    (unless (member word $inclAssertions)))
```
(setq |$inclAssertions| (cons word |$inclAssertions|))))

---

defun xlIfSyntax
[Top? p304]
[Else? p305]
[xlMsg p311]
[inclmsgIfSyntax p322]

--- defun xlIfSyntax ---

defun |xlIfSyntax| (eb str lno ufos info sts)
(let (context found st)
 (setq st (elt sts 0))
 (setq found (elt info 2))
 (setq context
   (cond
     ((|Top?| st) '|not in an )if...)endif|)
     ((|Else?| st) '|after an )else|)
     (t '|but can’t figure out where|)))
 (|xlMsg| eb str lno (elt ufos 0)
   (list (|inclmsgIfSyntax| (elt ufos 0) found context) '|error|))))

---

defun inclmsgIfSyntax
[concat p1197]
[theid p315]
[theorigin p313]

--- defun inclmsgIfSyntax ---

defun |inclmsgIfSyntax| (ufo found context)
(let (found (concat "" found))
 (list
   (format nil
     "Incorrect )if...)endif syntax. A %1f was found %2f. ~
       The processing of the source from %3f has been abandoned.")
   (list (|theid| found) (|theid| context) (|theorigin| ufo))))

---
defun xIfBug

(defun xIfBug (eb str lno ufos)
  (xMsg eb str lno (elt ufos 0) (list (inclmsgIfBug) 'bug)))

defun inclmsgIfBug

(defun inclmsgIfBug ()
  (list "Unexpected state in )if...endif." nil))

defun xCmdBug

(defun xCmdBug (eb str lno ufos)
  (xMsg eb str lno (elt ufos 0) (list (inclmsgCmdBug) 'bug)))

defun inclmsgCmdBug

(defun inclmsgCmdBug ()
  (list "Unexpected command in source inclusion." nil))
defvar incCommands

This is a list of commands that can be in an include file

— postvars —

eval-when (eval load)
(setq incCommands
(list "say" "include" "console" "fin" "assert" "if" "elseif" "else" "endif")
)

defvar $pfMacros

The $pfMacros variable is an alist [[id, state, body-pform], ...] where state is one of: mbody, mparam, mlambda

User-defined macros are maintained in a stack of definitions. This is the stack sequence resulting from the command lines:

a => 3
a => 4
b => 7

( (b |mbody| ((integer| (posn| (0 "b => 7" 1 i "strings") . 6)) . "7"))
 (a |mbody| ((integer| (posn| (0 "a => 4" 1 i "strings") . 6)) . "4"))
 (a |mbody| ((integer| (posn| (0 "a => 3" 1 i "strings") . 6)) . "3"))
 )

— initvars —

(defvar $pfMacros nil)

—

defun incClassify

; incClassify(s) ==
; not incCommand? s => [false,0,""]
; i := 1; n := #s
; while i < n and s.i = char " " repeat i := i + 1
; i >= n => [true,0,"other"]
; eb := (i = 1 => 0; i)
; bad:=true
; for p in incCommands while bad repeat
11.3. HELPER FUNCTIONS

; incPrefix?(p, i, s) =>
;   bad:=false
;   p1 := p
;   if bad then [true,0,"other"] else [true,eb,p1]

[incCommand? p326]
[incCommands p324]

— defun incClassify —

(defun incClassify (s)
  (let ((p1 bad eb n i)
    (declare (special incCommands)))
    (if (null (incCommand? s))
      (list nil 0 "")
    (progn
      (setq i 1)
      (setq n (length s))
      ((lambda ()
         (loop
           (cond
             ((not (and (< i n) (char= (elt s i) #\space)))
              (return nil))
             (t (setq i (1+ i))))))
      (cond
        ((not (< i n)) (list t 0 "other"))
      (t
        (if (= i 1)
          (setq eb 0)
          (setq eb i))
        (setq bad t)
        ((lambda (tmp1 p)
           (loop
             (cond
               ((or (atom tmp1)
                 (progn (setq p (car tmp1)) nil)
                (not bad))
               (return nil))
             (t
               (cond
                 ((incPrefix? p i s)
                  (identity
                   (progn
                     (setq bad nil)
                     (setq p1 p)))))))
             (setq tmp1 (cdr tmp1))))
      incCommands nil)
      (if bad
        (list t 0 "other"))))
defun incCommand?

incCommand? : String → Boolean
— defun incCommand? 0 —

(defun incCommand? (s)
  "does this start with a close paren?"
  (and (< 0 (length s)) (equal (elt s 0) #\)) )

——

defun incPrefix?

;incPrefix?(prefix, start, whole) ==
;  #prefix > #whole-start => false
;  good:=true
;  for i in 0..#prefix-1 for j in start.. while good repeat
;    good:= prefix.i = whole.j
;  good

— defun incPrefix? 0 —

(defun incPrefix? (prefix start whole)
  (let (good)
    (cond
      ((< (- (length whole) start) (length prefix)) nil)
      (t
        (setq good t)
        (lambda (Var i j)
          (loop
            (cond
              ((or (> i Var) (not good)) (return nil))
              (t (setq good (equal (elt prefix i) (elt whole j))))
              (setq i (+ i 1))
              (setq j (+ j 1))))
            (- (length prefix) 1) 0 start)
        good))))

——
defun incCommandTail

[incDrop p327]

— defun incCommandTail —

(defun incCommandTail (s info)
  (let ((start (elt info 1)))
    (when (= start 0) (setq start 1))
    (incDrop (+ start (length (elt info 2)) 1) s)))

---

defun incDrop

[substring p256]

— defun incDrop 0 —

(defun incDrop (n b)
  (if (>= n (length b))
    '|
    (substring b n nil)))

---

defun inclFname

[incFileName p862]
[incCommandTail p327]

— defun inclFname —

(defun inclFname (s info)
  (inclFileName (incCommandTail s info)))

---

defun incFileInput

[incRgen p329]
[make-instream p1127]

— defun incFileInput —
(defun incFileInput (fn)
  (incRgen (make-instream fn)))

(defun incConsoleInput

  (defun incConsoleInput ()
    (incRgen (make-instream 0)))

(defun incNConsoles

  (defun incNConsoles (ufos)
    (let ((a (member "console" ufos)))
      (if a
        (+ 1 (incNConsoles (cdr a)))
        0)))

(defun incActive?

  (defun incActive? (fn ufos)
    (member fn ufos))
11.3. HELPER FUNCTIONS

defun incRgen

Note that incRgen1 recursively calls this function.

(defun |incRgen| (s)
  (|Delay| #'|incRgen1| (list s)))

---

defun Delay

Delay prepends a label nonnullstream, returning a list of the label, the given function name in function and arguments. That is, given

(|Delay| |incLude1| (0 ("1") 0 ("strings") (1)))

construct

(|nonnullstream| |incLude1| 0 ("1") 0 ("strings") (1))

Note that nonnullstream is NOT a function so the inputs have been changed from a function call to a simple list.

Delay : (Function,List(Any)) \rightarrow Delay

— defun Delay 0 —

(defun |Delay| (function arguments)
  (cons '|nonnullstream| (cons function arguments)))

---

defvar StreamNil

— initvars —

(defvar |StreamNil| (list 'nullstream))

---

— postvars —
Chapter 11. Handling Terminal Input

(defun incRgen1)
This function reads a line from the stream and then conses it up with a recursive call to incRgen. Note that incRgen recursively wraps this function in a delay list.

```
(eval-when (eval load)
  (setq |StreamNil| (list '|nullstream|)))
```

```
defun incRgen1
  (defun incRgen1 (&rest z)
    (let (a s)
      (declare (special |StreamNil|))
      (setq s (car z))
      (setq a (read-line s nil nil))
      (if (null a)
        (progn
          (close s)
          |StreamNil|)
        (cons a (incRgen s)))))))
```

Chapter 12

The Token Scanner

defvar scanKeyWords

— postvars —

(eval-when (eval load)
defvar [scanKeyWords]
(list
 (list "add" 'add)
 (list "and" 'and)
 (list "break" 'break)
 (list "by" 'by)
 (list "case" 'case)
 (list "default" 'default)
 (list "define" 'defn)
 (list "do" 'do)
 (list "else" 'else)
 (list "exit" 'exit)
 (list "export" 'export)
 (list "for" 'for)
 (list "free" 'free)
 (list "from" 'from)
 (list "has" 'has)
 (list "if" 'if)
 (list "import" 'import)
 (list "in" 'in)
 (list "inline" 'inline)
 (list "is" 'is)
 (list "isnt" 'isnt)
 (list "iterate" 'iterate)
 (list "local" '|local|)
 (list "macro" 'macro)
(list "mod" 'mod)
(list "or" 'or)
(list "pretend" 'pretend)
(list "quo" 'quo)
(list "rem" 'rem)
(list "repeat" 'repeat)
(list "return" 'return)
(list "rule" 'rule)
(list "then" 'then)
(list "where" 'where)
(list "while" 'while)
(list "with" 'with)
(list ";" 'dot)
(list ":-" 'coerce)
(list ":" 'colon)
(list ":==" 'colondash)
(list ":@" 'at)
(list ":@@" 'atat)
(list ":," 'comma)
(list ":;" 'semicolon)
(list ":**" 'power)
(list ":*" 'times)
(list ":+" 'plus)
(list ":-" 'minus)
(list ":<" 'lt)
(list ":>" 'gt)
(list ":<<" 'le)
(list ":>>" 'ge)
(list ":=` 'equal)
(list ":~=" 'notequal)
(list ":~" '~~)
(list ":^" 'carat)
(list ":.." 'seg)
(list ":#" '#!)
(list ":&" 'ampersand)
(list ":$" '$)
(list ":/" 'slash)
(list ":\" 'backslash)
(list "://" 'slasheslash)
(list ":\\\" 'backslashbackslash)
(list ":\\/" 'slashbackslash)
(list ":\\\" 'backslashslash)
(list ":=>" 'exit)
(list "::=" 'becomes)
(list ":==" 'def)
(list ":==>" 'mdef)
(list ":->" 'arrow)
(list ":<-" 'larrow)
(list ":+-" 'gives)
defvar infgeneric

— postvars —

(eval-when (eval load)
(prog ()
  (return
   ((lambda (var value)
      (loop
       (cond
       ((or (atom var) (progn (setq value (car var)) nil))
        (return nil))
      (t
        (setf (get (car value) 'infgeneric) (cadr value))))
      (setq var (cdr var))))
    (list (list 'equal '=)
          (list 'times '*)
          (list 'has '|has|)
          (list 'case '|case|)
          (list 'rem '|rem|)
          (list 'mod '|mod|)
          (list 'quo '|quo|)
          (list 'slash '/)
defun lineoftoks

lineoftoks bites off a token-dq from a line-stream returning the token-dq and the rest of the line-stream

`;lineoftoks(s)==
 ; $f: local:=nil
 ; $r:local :=nil
 ; $ln:local :=nil
 ; $linepos:local:=nil
 ; $n:local:=nil
 ; $sz:local := nil
 ; $floatok:local:=true
 ; if not nextline s
 ; then CONS(nil,nil)
 ; else
 ; if null scanIgnoreLine($ln,$n) -- line of spaces or starts ) or >
 ; then cons(nil,$r)
 ; else
toks:=[]
a:= incPrefix?("command",1,$ln)
a =>
 ; $ln:=SUBSTRING($ln,8,nil)
b := dqUnit constoken($ln,$linepos,"command","ln");
cons([ [b,s] ],$r)
while $n<$sz repeat toks:=dqAppend(toks,scanToken())
if null toks
then cons([],$r)
else cons([ [toks,s] ],$r)

— defun lineoftoks —
(defun lineoftoks (s)
(let ((floatok $sz $n $linepos $ln $r $f b a toks))
declare (special floatok $f $sz $linepos $r $n $ln))
(setq $f nil)
(setq $r nil)
(setq $ln nil)
(setq $linepos nil)
(setq $n nil)
(setq $sz nil)
(setq $floatok t)
(cond
((null (nextline s)) (cons nil nil))
((null (scanIgnoreLine $ln $n)) (cons nil $r))
(t
(setq toks nil)
(setq a (incPrefix? "command" 1 $ln))
cond
(a
(setq $ln (substring $ln 8 nil))
(setq b)
(dqUnit (consonstoken $ln $linepos (list 'command "$ln") 0))
(cons (list (list b s)) $r))
(t
((lambda ()
  loop

[nextline p336]
[scanIgnoreLine p336]
[incPrefix? p326]
[substring p256]
[dqUnit p565]
[constoken p337]
[floatok p??]
[f p??]
[sz p??]
[linepos p??]
[r p??]
[n p??]
[ln p??]

336]
defun nextline

(defvar sz n linepos ln r f)

(defun nextline (s)
  (declare (special sz n linepos ln r f))
  (cond
   ((npNull s) nil)
   (t
    (setq f (car s))
    (setq r (cdr s))
    (setq ln (cdr f))
    (setq linepos (caar f))
    (setq n (strposl " " ln 0 t)); spaces at beginning
    (setq sz (length ln))
    t))
)

---

defun scanIgnoreLine

(defvar incPrefix)

(defun scanIgnoreLine (ln n)
(cond
  ((null n) n)
  (t
   (cond
     (eql (char-code (char ln 0)) (char-code #\))
     (cond
       (inPrefix? "command" 1 ln) t
       (t nil)))))
   (t n))))

---

defun constoken

(ncPutQ p638)

---

defun constoken ---

(defun constoken (ln lp b n)
  (declare (ignore ln))
  (let (a)
    (setq a (cons (elt b 0) (elt b 1)))
    (ncPutQ a ’posn (cons lp n))
    a))

---

defun scanToken

[startsComment? p339]
[scanComment p339]
[startsNegComment? p340]
[scanNegComment p340]
[lid p338]
[punctuation? p341]
[scanPunct p341]
[startsId? p1194]
[scanWord p349]
[scanSpace p352]
[scanString p353]
[scanNumber p355]
[scanEscape p358]
[scanError p358]
[dqUnit p565]
defun scanToken

(let (b ch n linepos c ln)
  (declare (special $linepos $n $ln))
  (setq ln $ln)
  (setq c (char-code (char $ln $n)))
  (setq linepos $linepos)
  (setq n $n)
  (setq ch (elt $ln $n))
  (setq b
    (cond
      ((startsComment?) (scanComment) nil)
      ((startsNegComment?) (scanNegComment) nil)
      ((= c (char-code \\?))
        (setq n (+ n 1))
        (lfid "?
      )
      ((punctuation? c) (scanPunct))
      ((startsId? ch) (scanWord) nil)
      ((= c (char-code \space)) (scanSpace) nil)
      ((= c (char-code \"))) (scanString)
      (digitp ch) (scanNumber))
      (t (scanError)))))
  (cond
    ((null b) nil)
    (t
      (dqUnit
        (constoken ln linepos b (+ n (lnExtraBlanks linepos)))))))

---

defun lfid

To pair badge and badgee

--- defun lfid 0 ---

(defun lfid (x)
  (list 'id (intern x "BOOT")))

---
defun Is it a ++ comment?

(defun startsComment? 0 |
  (defun |startsComment?| ()
    (let (www)
      (declare (special |$ln| |$sz| |$n|))
      (cond
        ((< |$n| |$sz|)
         (cond
          ( (= (char-code (char |$ln| |$n|)) (char-code #\+))
           (setq www (+ |$n| 1)))
          (cond
           (not (< www |$sz|)) nil)
           (t (= (char-code (char |$ln| www)) (char-code #\+))))))
      (t nil)))
    (t nil))))

defun scanComment

(defun scanComment ()
  (let (n)
    (declare (special |$ln| |$sz| |$n|))
    (setq n |$n|)
    (setq |$n| |$sz|)
    ((|lfcomment| (substring |$ln| n nil))))

———

(defun Is it a ++ comment?

(defun startsComment? 0 |
  (defun |startsComment?| ()
    (let (www)
      (declare (special |$ln| |$sz| |$n|))
      (cond
        ((< |$n| |$sz|)
         (cond
          ( (= (char-code (char |$ln| |$n|)) (char-code #\+))
           (setq www (+ |$n| 1)))
          (cond
           (not (< www |$sz|)) nil)
           (t (= (char-code (char |$ln| www)) (char-code #\+))))))
      (t nil)))
    (t nil))))

defun scanComment

(defun scanComment ()
  (let (n)
    (declare (special |$ln| |$sz| |$n|))
    (setq n |$n|)
    (setq |$n| |$sz|)
    ((|lfcomment| (substring |$ln| n nil))))

———
defun lfcomment

--- defun lfcomment 0 ---

(defun lfcomment (x)
  (list '|comment| x))

---

defun Is it a comment?

([$ln p??]
 [$sz p??]
 [$n p??]

--- defun startsNegComment? ---

(defun startsNegComment? ()
  (let (www)
    (declare (special |$ln| |$sz| |$n|))
    (cond
      ((< |$n| |$sz|)
        (cond
          ((= (char-code (char |$ln| |$n|)) (char-code #\-))
           (setq www (+ |$n| 1))
           (cond
            ((not (< www |$sz|)) nil)
            (t (= (char-code (char |$ln| www)) (char-code #\-))))
          (t nil))
        (t nil))))

---

defun scanNegComment

[lfnegcomment p341]
[substring p256]
[$ln p??]
[$sz p??]
[$n p??]

--- defun scanNegComment ---
(defun |scanNegComment| ()
  (let (n)
    (declare (special |$ln| |$sz| |$n|))
    (setq n |$n|)
    (setq |$n| |$sz|)
    (|lfnegcomment| (substring |$ln| n nil)))))

defun lfnegcomment
  — defun lfnegcomment 0 —

  (defun |lfnegcomment| (x)
    (list '|negcomment| x))

defun punctuation?
  — defun punctuation? —

  (defun |punctuation?| (c)
    (eql (elt |scanPun| c) 1))

defun scanPunct
  [subMatch p342]
  [scanError p358]
  [scanKeyTr p343]
  [$n p??]
  [$ln p??]

  — defun scanPunct —

  (defun |scanPunct| ()
    (let (a sss)
      (declare (special |$n| |$ln|)))
(setq sss (subMatch |$ln| |$n|))
(setq a (length sss))
(cond
  ((eql a 0) (scanError))
  (t (setq |$n| (+ |$n| a)) (scanKeyTr sss))))

defun subMatch

(defun subMatch (a b)
  (substringMatch a |scanDict| b))

defun substringMatch

;substringMatch (l,d,i)==
; h:= QENUM(l, i)
; u:=ELT(d,h)
; ll:=SIZE l
; done:=false
; s1:=""
; for j in 0.. SIZE u - 1 while not done repeat
; s:=ELT(u,j)
; ls:=SIZE s
; done:=if ls+i > ll
; then false
; else
;   eql:= true
;   for k in 1..ls-1 while eql repeat
;     eql:= EQL(QENUM(s,k),QENUM(l,k+i))
;   if eql
;     then
;       s1:=s
;       true
;     else false
;   s1

— defun substringMatch —
(defun |substringMatch| (l dict i)
  (let (equl ls s s1 done ll u h)
    (setq h (char-code (char l i)))
    (setq u (elt dict h))
    (setq ll (size l))
    (setq s1 "")
    ((lambda (Var4 j)
      (loop
        (cond
          ((or (> j Var4) done) (return nil))
          (t
            (setq s (elt u j))
            (setq ls (size s))
            (setq done
              (cond
                ((< ll (+ ls i)) nil)
                (t
                  (setq equl t)
                  ((lambda (Var5 k)
                    (loop
                      (cond
                        ((or (> k Var5) (not equl)) (return nil))
                        (t
                          (setq equl (= (char-code (char s k))
                                        (char-code (char l (+ k i)))))
                          (setq equl t)
                          (setq k (+ k 1)))
                        (- ls 1) 1)
                      (cond (equl (setq s1 s) t) (t nil)))))))
            (setq j (+ j 1))))
        (- (size u) i) 0)
      s1))

---

defun scanKeyTr

[keyword p344]
[scanPossFloat p344]
[$key p345]
[scanCloser? p349]
[$f|loatok p??]

— defun scanKeyTr —

(defun |scanKeyTr| (w)
  (declare (special |$f|loatok))
  (cond
((eq (keyword w) 'dot)
  (cond
   (|$floatok| (|scanPossFloat| w))
   (t (|lfkey| w)))
  (t (setq |$floatok| (null (|scanCloser?| w))) (|lfkey| w))))

---

defun keyword

[|hget p1194|]

---

— defun keyword 0 —

(defun |keyword| (st)
  (hget |scanKeyTable| st))

---

defun keyword?

[|hget p1194|]

---

— defun keyword? 0 —

(defun |keyword?| (st)
  (null (null (hget |scanKeyTable| st))))

---

defun scanPossFloat

[|lfkey p345|]
[|spleI p345|]
[|scanExponent p349|]
[|$ln p??|]
[|$sz p??|]
[|$n p??|]

— defun scanPossFloat —

(defun |scanPossFloat| (w)
(declare (special ln sz n))

(cond
  ((or (not (< n sz)) (null (digitp (elt ln n))))
   (lfkey w))
  (t
   (setq w (spleI #'digitp) (scanExponent "0" w))))

---

defun digit?

[digitp p1195]

---

---

defun lfkey

[keyword p344]

---

---

defun spleI

[spleI p346]

---
defun spleI1

[substring p256]
[scanEsc p346]
[spleI1 p346]
[concat p1197]
[$ln p??]
[$sz p??]
[$n p??]

— defun spleI1 —

(defun |spleI1| (dig zro)
(let (bb a str l n)
  (declare (special $ln $sz $n))
  (setq n $n)
  (setq l $sz)
  ; while $n<l and FUNCALL(dig,(ln.$n)) repeat $n:=$n+1
  ((lambda ()
     (loop
       (cond
         ((not (and (< $n l) (funcall dig (elt $ln $n))))
          (return nil))
         (t
          (setq $n (+ $n 1)))))))
  (cond
    ((or (equal $n l) (not (= (char-code (char $ln $n)) (char-code #\_))))
     (cond
      ((equal n $n) zro) "0")
      (t (substring $ln n (- $n n))))
    (t ; escaped
     (setq str (substring $ln n (- $n n)))
     (setq $n (+ $n 1))
     (setq a (|scanEsc|))
     (setq bb (|spleI1| dig zro)) ; escape, any number of spaces are ignored
     (concat str bb))))

defun scanEsc

;scanEsc()==
; if $n>=$sz
; then if nextline($r)
; then
; while null $n repeat nextline($r)
; scanEsc()
false
else false
else
n1:=STRPOSL(" ", $ln, $n, true)
if null n1
then if nextline($r)
then
while null $n repeat nextline($r)
scanEsc()
false
else false
else
if $n=n1
then true
else if QENUM($ln, n1)=ESCAPE
then
$n:=n1+1
scanEsc()
false
else
$n:=n1
startsNegComment?() or startsComment?() =>
nextline($r)
scanEsc()
false
false
[newline p336]
[scanEsc p346]
[strpos p1196]
[startsNegComment? p340]
[startsComment? p339]
[$ln p?]
[$r p?]
[$sz p?]
[$n p?]

— defun scanEsc —

(defun |scanEsc| ()
(let (n1)
(declare (special |$ln| |$r| |$sz| |$n|))
(cond
((not (< |$n| |$sz|))
(cond
((null |nextline| |$r|)
((lambda ()
(loop
(cond


defvar scanCloser

— postvars —

(eval-when (eval load)
  (defvar |scanCloser| (list '|) |} ']' '\|\| '\|\| '\|\|)))
defun scanCloser?

[keyword p344]
[scanCloser p348]

— defun scanCloser? 0 —

(defun |scanCloser?| (w)
  (declare (special |scanCloser|))
  (member (|keyword| w) |scanCloser|))

------

defun scanWord

[scanW p351]
[lfid p338]
[keyword? p344]
[lfkey p345]
[$floatok p??]

— defun scanWord —

(defun |scanWord| (esp)
  (let (w aaa)
    (declare (special |$floatok|))
    (setq aaa (|scanW| nil))
    (setq w (elt aaa 1))
    (setq |$floatok| nil)
    (cond
      ((or esp (elt aaa 0))
       (|lfid| w))
      ((|keyword?| w)
       (setq |$floatok| t)
       (|lfkey| w))
      (t
       (|lfid| w)))))

------

defun scanExponent

[lffloat p351]
digit? p345]
defun scanExponent (a w)
(let (c1 e c n)
 (declare (special $ln $sz $n))
 (cond
 (not (< $n $sz)) (lffloat a w "0")
 (t
 (setq n $n)
 (setq c (char-code (char $ln $n)))
 (cond
 ((or (= c (char-code #\E)) (= c (char-code #\e)))
 (setq $n (+ $n 1))
 (cond
 ((not (< $n $sz))
 (setq $n n)
 (lffloat a w "0")
 ((digitp (elt $ln $n))
 (setq e (spleI #’digitp))
 (lffloat a w e))
 (t
 (setq c1 (char-code (char $ln $n)))
 (cond
 ((or (= c1 (char-code #\+)) (= c1 (char-code #\-)))
 (setq $n (+ $n 1))
 (cond
 ((not (< $n $sz))
 (setq $n n)
 (lffloat a w "0")
 ((digitp (elt $ln $n))
 (setq e (spleI #’digitp))
 (lffloat a w
 (cond
 (=(c1 (char-code #\-))
 (concat "-" e))
 (t e))))
 (t
 (setq $n n)
 (lffloat a w "0").))))))
 (t (lffloat a w "0"))))))
defun lffloat

[concat p1197]

— defun lffloat 0 —

(defun lffloat (a w e)
  (list \"float\" (concat a "." w "e" e)))

———

defmacro idChar?

— defmacro idChar? 0 —

(defmacro idChar? (x)
  `(or (alphanumericp ,x) (member ,x '(#\? #\% #\' #\!) :test #'char=))))

———

defun scanW

[posend p352]
[substring p256]
[scanEsc p346]
[scanW p351]
[idChar? p351]
[concat p1197]
[$ln p??]
[$sz p??]
[$n p??]

— defun scanW —

(defun scanW (b)
  (let (bb a strendid 1 n1)
    (declare (special |$ln| |$sz| |$n|))
    (setq n1 |$n|)
    (setq |$n| (+ |$n| 1))
    (setq 1 |$sz|)
    (setq endid (|posend| |$ln| |$n|))
    (cond
      ((or (equal endid 1)
(not (= (char-code (char |$ln| endid)) (char-code #\_))))
(setq |$n| endid)
(list b (substring |$ln| n1 (- endid n1))))
(t
(setq str (substring |$ln| n1 (- endid n1)))
(setq |$n| (+ endid 1))
(setq a (lscanEsc))
(setq bb
(cond
  (a (lscanW t))
  ((not (< |$n| |$sz|)) (list b ""))
  (((idChar? (elt |$ln| |$n|)) (lscanW b))
    (t (list b ""))))
  (list (or (elt bb 0) b) (concat str (elt bb 1))))))

defun posend

; posend(line,n)==
;   while n<#line and idChar? line.n repeat n:=n+1
;   n

NOTE: do not replace "lyne" with "line"

— defun posend —

(defun |posend| (lyne n)
  (lambda ()
    (loop
      (cond
        ((not (and (< n (length lyne)) (idChar? (elt lyne n))))
         (return nil))
        (t (setq n (+ n 1))))))))

——

defun scanSpace

[strpos p1196]
[lspaces p353]
[$floatok p??]
[$ln p??]
[$n p??]

— defun scanSpace —
(defun |scanSpace| ()
  (let (n)
    (declare (special |$floatok| |$ln| |$n|))
    (setq n |$n|)
    (setq |$n| (strposl " " |$ln| |$n| t))
    (when (null |$n|) (setq |$n| (length |$ln|)))
    (setq |$floatok| t)
    (|lfspaces| (- |$n| n))))

---

defun lfspaces

defun lfspaces 0 —

(defun |lfspaces| (x)
  (list '|spaces| x))

---

defun scanString

[lfstring p353]
[scanS p354]
[$floatok p??]
[$n p??]

— defun scanString —

(defun |scanString| ()
  (declare (special |$floatok| |$n|))
  (setq |$n| (+ |$n| 1))
  (setq |$n| (+ |$n| i))
  (setq |$floatok| nil)
  (|lfstring| (|scanS|)))

---

defun lfstring

— defun lfstring 0 —
(defun lfstring (x)
  (if (eql (length x) 1)
      (list |char| x)
      (list |string| x)))

---

defun scanS

(ncSoftError p574)
(lnExtraBlanks p567)
(strpos p196)
(substring p256)
(scanEsc p346)
(concat p197)
(scanTransform p355)
(scanS p354)
(ln p??)
(linepos p??)
(sz p??)
(n p??)

--- defun scanS ---

(defun scanS ()
  (let (b a str mn escsym strsym n)
    (declare (special $ln |$linepos| $sz |$n|))
    (cond
      ((not (< |$n| $sz))
       (ncSoftError)
       (cons $linepos (+ (lnExtraBlanks |$linepos|) |$n|))
       "Quote added at end of line." nil) "")
    (t
      (setq n |$n|)
      (setq strsym (or (strpos "" |$ln| |$n| nil) $sz))
      (setq escsym (or (strpos _ |$ln| |$n| nil) $sz))
      (setq mn (min strsym escsym))
      (cond
       ((equal mn $sz)
        (setq |$n| $sz))
       (ncSoftError)
       (cons $linepos (+ (lnExtraBlanks |$linepos|) |$n|))
       "Quote added at end of line." nil)
       (substring |$ln| n nil))
      ((equal mn strsym)
       (setq |$n| (+ mn 1))
       (substring |$ln| n (- mn n))))

---

defun scanS

(ncSoftError p574)
(lnExtraBlanks p567)
(strpos p196)
(substring p256)
(scanEsc p346)
(concat p197)
(scanTransform p355)
(scanS p354)
(ln p??)
(linepos p??)
(sz p??)
(n p??)

--- defun scanS ---

(defun scanS ()
  (let (b a str mn escsym strsym n)
    (declare (special $ln |$linepos| $sz |$n|))
    (cond
      ((not (< |$n| $sz))
       (ncSoftError)
       (cons $linepos (+ (lnExtraBlanks |$linepos|) |$n|))
       "Quote added at end of line." nil) "")
    (t
      (setq n |$n|)
      (setq strsym (or (strpos "" |$ln| |$n| nil) $sz))
      (setq escsym (or (strpos _ |$ln| |$n| nil) $sz))
      (setq mn (min strsym escsym))
      (cond
       ((equal mn $sz)
        (setq |$n| $sz))
       (ncSoftError)
       (cons $linepos (+ (lnExtraBlanks |$linepos|) |$n|))
       "Quote added at end of line." nil)
       (substring |$ln| n nil))
      ((equal mn strsym)
       (setq |$n| (+ mn 1))
       (substring |$ln| n (- mn n))))

---

defun scanS

(ncSoftError p574)
(lnExtraBlanks p567)
(strpos p196)
(substring p256)
(scanEsc p346)
(concat p197)
(scanTransform p355)
(scanS p354)
(ln p??)
(linepos p??)
(sz p??)
(n p??)

--- defun scanS ---

(defun scanS ()
  (let (b a str mn escsym strsym n)
    (declare (special $ln |$linepos| $sz |$n|))
    (cond
      ((not (< |$n| $sz))
       (ncSoftError)
       (cons $linepos (+ (lnExtraBlanks |$linepos|) |$n|))
       "Quote added at end of line." nil) "")
    (t
      (setq n |$n|)
      (setq strsym (or (strpos "" |$ln| |$n| nil) $sz))
      (setq escsym (or (strpos _ |$ln| |$n| nil) $sz))
      (setq mn (min strsym escsym))
      (cond
       ((equal mn $sz)
        (setq |$n| $sz))
       (ncSoftError)
       (cons $linepos (+ (lnExtraBlanks |$linepos|) |$n|))
       "Quote added at end of line." nil)
       (substring |$ln| n nil))
      ((equal mn strsym)
       (setq |$n| (+ mn 1))
       (substring |$ln| n (- mn n))))
(setq str (substring |$ln| n (- mn n)))
(setq |$n| (+ mn 1))
(setq a (|scanEsc|))
(setq b (cond
    (a
        (setq str (concat str (|scanTransform| (elt |$ln| |$n|))))
        (setq |$n| (+ |$n| 1)) (|scanS|))
    (t (|scanS|))))
(concat str b))))))

______

defun scanTransform

    — defun scanTransform —

(defun |scanTransform| (x) x)

______

defun scanNumber

[spleI p345]
[linteger p357]
spleI p346
[scanExponent p349]
[scanCheckRadix p357]
lfrinteger p357
[concat p1197]
[$floatok p??]
[$ln p??]
[$sz p??]
[$n p??]

    — defun scanNumber —

(defun |scanNumber| ()
(let (v w n a)
  (declare (special |$floatok| |$ln| |$sz| |$n|))
  (setq a (|spleI| #'digitp))
  (cond
(not (< |$n| |$sz|))
  (|lfinteger| a))
  (not (= (char-code (char |$ln| |$n|)) (char-code #\r)))
  (cond
    ((and |$floatok| (= (char-code (char |$ln| |$n|)) (char-code #\.)))
     (setq n |$n|)
     (setq |$n| (+ |$n| 1)))
    (cond
      ((and (< |$n| |$sz|) (= (char-code (char |$ln| |$n|)) (char-code #\.)))
       (setq |$n| n)
       (|lfinteger| a))
      (t
       (setq w (|spleI1| #'digitp t))
       (|scanExponent| a w)))
    (t (|lfinteger| a))))
(t
  (setq |$n| (+ |$n| 1))
  (setq w (|spleI1| '#|rdigit?| t))
  (|scanCheckRadix| (parse-integer a) w)
  (cond
    ((not (< |$n| |$sz|))
     (|lfinteger| a w))
    ((= (char-code (char |$ln| |$n|)) (char-code #\.))
     (setq n |$n|)
     (setq |$n| (+ |$n| 1)))
    (cond
      ((and (< |$n| |$sz|) (= (char-code (char |$ln| |$n|)) (char-code #\.)))
       (setq |$n| n)
       (|lfinteger| a w))
      (t
       (setq v (|spleI1| '#|rdigit?| t))
       (|scanCheckRadix| (parse-integer a) v)
       (|scanExponent| (concat a "r" w) v))
      (t (|lfinteger| a w))))))

---

defun rdigit?

[strpos p1196]

— defun rdigit? 0 —

(defun |rdigit?| (x)
  (strpos x "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ" 0 nil))

---
defun linteger

--- defun linteger 0 ---

(defun linteger (x) (list 'integer x))

----------

defun lfrinteger

[concat p1197]

--- defun lfrinteger 0 ---

(defun lfrinteger (r x) (list 'integer (concat r (concat "r" x))))

----------

defun scanCheckRadix

; scanCheckRadix(r,w)==
; ns:=#w
; done:=false
; for i in 0..ns-1 repeat
; a:=rdigit? w.i
; if null a or a>=r
; then ncSoftError(cons($linepos,lnExtraBlanks $linepos+$n-ns+i),
; "S2CN0002", [w.i])

[$n p??]
[|linepos p??|

--- defun scanCheckRadix ---

(defun scanCheckRadix (r w) (let (a ns) (declare (special $n |$linepos|)) (setq ns (length w)) ((lambda (Var1 i) (loop (cond
<i>CHAPTER 12. THE TOKEN SCANNER</i>

```lisp
((> i Var1) (return nil))
(t
  (setq a (|rdigit?| (elt w i)))
  (cond
    ((or (null a) (not (< a r)))
     (|ncSoftError|
      (cons $linepos| (+ (- (+ (|lnExtraBlanks| $linepos|) $n|) ns) i))
      "The character %1 is greater than the radix."
      (list (elt w i))))))
  (setq i (+ i 1))))
(- ns 1) 0)))

defun scanEscape

[scanEsc p346]
[scanWord p349]
[$n p??]

— defun scanEscape —

(defun |scanEsc| ()
  (declare (special |$n|))
  (setq |$n| (+ |$n| 1))
  (when (|scanEsc|) (|scanWord| t)))

— defun scanError —

defun scanError

[ncSoftError p574]
[lnExtraBlanks p567]
[lferror p359]
[$ln p??]
[$linepos p??]
[$n p??]

(defun |scanError| ()
  (let (n)
    (declare (special |$ln| |$linepos| |$n|))
    (setq n |$n|))
(setq $n (+ $n 1))
(incSoftError)
(cons $linepos (+ (lnExtraBlanks $linepos) $n))
"The character %1 is not an AXIOM character."
(list (elt $ln n)))
(1ferror (elt $ln n))))

---

defun lferror

— defun lferror 0 —

(defun lferror (x)
 (list *error* x))

---

defvar scanKeyTable

— postvars —

(eval-when (eval load)
 (defvar scanKeyTable (scanKeyTableCons)))

---

defun scanKeyTableCons

This function is used to build the scanKeyTable

;scanKeyTableCons()==
; KeyTable:=MAKE_-HASHTABLE("CVEC",true)
; for st in scanKeyWords repeat
; HPUT(KeyTable,CAR st,CADR st)
; KeyTable

— defun scanKeyTableCons —
(defun |scanKeyTableCons| ()
  (let (KeyTable)
    (setq KeyTable (make-hash-table :test #'equal))
    ((lambda (Var6 st)
        (loop
          (cond
            ((or (atom Var6) (progn (setq st (car Var6)) nil))
              (return nil))
            (t
              (hput KeyTable (car st) (cadr st))))
        (setq Var6 (cdr Var6))))
      |scanKeyWords| nil)
    KeyTable))

defvar scanDict

  — postvars —

(eval-when (eval load)
  (defvar |scanDict| (|scanDictCons|)))

defun scanDictCons

  ; scanDictCons()==
  ; l:= HKEYS scanKeyTable
  ; d :=
  ;   a:=MAKE_-VEC(256)
  ;   b:=MAKE_-VEC(1)
  ;   VEC_-SETELT(b,0,MAKE_-CVEC 0)
  ;   for i in 0..255 repeat VEC_-SETELT(a,i,b)
  ;   a
  ;   for s in l repeat scanInsert(s,d)
  ;   d

  [hkeys p1195]

  — defun scanDictCons —

  (defun |scanDictCons| ()
    (let (d b a l)
(setq l (hkeys |scanKeyTable|))
(setq d
(progn
 (setq a (make-array 256))
 (setq b (make-array 1))
 (setf (svref b 0)
 (make-array 0 :fill-pointer 0 :element-type 'string-char))
 ((lambda (i)
   (loop
    (cond
     ((> i 255) (return nil))
     (t (setf (svref a i) b))
     (setq i (+ i 1))))
   0)
   a))
 ((lambda (Var7 s)
   (loop
    (cond
     ((or (atom Var7) (progn (setq s (car Var7)) nil))
      (return nil))
     (t (|scanInsert| s d)))
    (setq Var7 (cdr Var7))))
 l nil)
 d))

---

(defun scanInsert

; scanInsert(s,d) == l := #s
; h := QENUM(s,0)
; u := ELT(d,h)
; n := #u
; k:=0
; while l <= #(ELT(u,k)) repeat
; \hspace{3mm} k:=k+1
; \hspace{10mm} v := MAKE_VEC(n+1)
; \hspace{10mm} for i in 0..k-1 repeat VEC_SETELT(v,i,ELT(u,i))
; \hspace{10mm} VEC_SETELT(v,k,s)
; \hspace{10mm} for i in k..n-1 repeat VEC_SETELT(v,i+1,ELT(u,i))
; \hspace{10mm} VEC_SETELT(d,h,v)
; l := s

---

(defun |scanInsert| (s d)
(let (v k n u h l)
  (setq l (length s))
  (setq h (char-code (char s 0)))
  (setq u (elt d h))
  (setq n (length u))
  (setq k 0)
  ((lambda ()
      (loop
        (cond
          ((< (length (elt u k)) l) (return nil))
          (t (setq k (+ k 1)))))))
  (setq v (make-array (+ n 1)))
  ((lambda (Var2 i)
      (loop
        (cond
          ((> i Var2) (return nil))
          (t (setf (svref v i) (elt u i))))
        (setq i (+ i 1))))
       (- k 1) 0)
  (setf (svref v k) s)
  ((lambda (Var3 i)
      (loop
        (cond
          ((> i Var3) (return nil))
          (t (setf (svref v (+ i 1)) (elt u i))))
        (setq i (+ i 1))))
       (- n 1) k)
  (setf (svref d h) v)
  s))

__________

defvar scanPun

— postvars —

(eval-when (eval load)
  (defvar |scanPun| (|scanPunCons|)))

__________

defun scanPunCons

;scanPunCons()==
; listing := HKEYS scanKeyTable
(defun scanPunCons ()
  (let (a listing)
    (setq listing (hkeys scanKeyTable))
    (setq a (make-array (list 256) :element-type 'bit :initial-element 0))
    ((lambda (i)
       (loop
         (cond
           ((> i 255) (return nil))
           (t (setf (sbit a i) 0)))
         (setq i (+ i 1))))
      0)
    ((lambda (Var8 k)
       (loop
         (cond
           ((or (atom Var8) (progn (setq k (car Var8)) nil))
            (return nil))
           (t
            (cond
              ((null (startsId? (elt k 0)))
               (setf (sbit a (char-code (char k 0))) 1)))
              (setq Var8 (cdr Var8))))
         listing nil)
      a))
    a))
Chapter 13

Input Stream Parser

defun Input Stream Parser

[trappoint p??]
[npFirstTok p367]
[npItem p366]
[ncSoftError p574]
[tokPosn p635]
[pfWrong p526]
[pfDocument p476]
[pfListOf p475]
[$ttok p??]
[$stok p??]
[$stack p??]
[$inputStream p??]

— defun npParse —

(defun npParse (stream)
  (let ((|$ttok| |$stok| |$stack| |$inputStream| found)
        (declare (special |$ttok| |$stack| |$inputStream| |$stok|))
        (setq |$inputStream| stream)
        (setq |$stack| nil)
        (setq |$stok| nil)
        (setq |$ttok| nil)
        (|npFirstTok|)
        (setq found (catch 'trappoint (|npItem|)))
        (cond
          ((eq found 'trapped)
           (|ncSoftError| (|tokPosn| |$stok|) "syntax error at top level" nil)
           (|pfWrong| (|pfDocument| "top level syntax error") (|pfListOf| nil)))
          ((null (null |$inputStream|))))

  365
(incSoftError (tokPosn $stok) "Improper syntax." nil)
(pfWrong
  (pfDocument (list "input stream not exhausted"))
  (pfList0f nil)))
(null $stack)
(incSoftError (tokPosn $stok)
  "System error while parsing, stack is empty." nil)
(pfWrong (pfDocument (list "stack empty") (pfList0f nil)))
(t (car $stack))))

---

(defun npItem ()
  (let ((c b a tmp1)
    (when (npQualDef)
      (if (npEqKey 'semicolon)
        (progn
          (setq tmp1 (npItem1 (npPop1)))
          (setq a (car tmp1))
          (setq b (cadr tmp1))
          (setq c (pfEnSequence b))
          (if a
            (npPush c)
            (npPush (pfNovalue c)))
          (npPush (pfEnSequence (npPop1))))))))

---

defun npItem1

[npQualDef p369]
[npEqKey p369]
[npItem1 p366]
[npPop1 p368]
[pfEnSequence p493]
[npPush p367]
[pfNovalue p509]

---

(defun npItem1 ()
  (let ((c b a tmp1)
    (when (npQualDef)
      (if (npEqKey 'semicolon)
        (progn
          (setq tmp1 (npItem1 (npPop1)))
          (setq a (car tmp1))
          (setq b (cadr tmp1))
          (setq c (pfEnSequence b))
          (if a
            (npPush c)
            (npPush (pfNovalue c)))
          (npPush (pfEnSequence (npPop1))))))))
— defun npItem1 —

(defun npItem1 (c)
  (let (b a tmp1)
    (if (npQualDef)
      (if (npEqKey 'semicolon)
       (progn
         (setq tmp1 (npItem1 (npPop1)))
         (setq a (car tmp1))
         (setq b (cadr tmp1))
         (list a (append c b)))
       (list t (append c (npPop1)))))
    (list nil c))))

— defun npFirstTok —

defun npFirstTok

Sets the current leaf ($stok) to the next leaf in the input stream. Sets the current token ($ttok) cdr of the leaf. A leaf token looks like [head, token, position] where head is either an id or (id . alist)

defun Push one item onto $stack

[$stack ??]

[[npPop1 p368]]
--- defun npPush 0 ---
(defun npPush (x)
  (declare (special $stack))
  (push x $stack))

---
defun Pop one item off $stack
[stack p??]
--- defun npPop1 0 ---
(defun npPop1 ()
  (declare (special $stack))
  (pop $stack))

---
defun Pop the second item off $stack
[stack p??]
--- defun npPop2 0 ---
(defun npPop2 ()
  (let (a)
    (declare (special $stack))
    (setq a (cadr $stack))
    (rplacd $stack (cddr $stack))
    a))

---
defun Pop the third item off $stack
[stack p??]
--- defun npPop3 0 ---
(defun npPop3 ()
  (defun npPop3 ()
(let (a)
  (declare (special |$stack|))
  (setq a (caddr |$stack|))
  (rplacd (cdr |$stack|) (cdddr |$stack|)) a))

---

defun npQualDef

[npComma p370]
[npPush p367]
[npPop1 p368]

— defun npQualDef —

(defun npQualDef ()
  (and (npComma) (npPush (list (npPop1))))))

---

defun Advance over a keyword

Test for the keyword, if found advance the token stream

[npNext p369]
[|$ttok p??]
[|$stok p??]

— defun npEqKey —

(defun npEqKey (keyword)
  (declare (special |$ttok| |$stok|))
  (and
   (eq (caar |$stok|) '|key|)
   (eq keyword |$ttok|)
   (npNext))))

---

defun Advance the input stream

This advances the input stream. The call to npFirstTok picks off the next token in the input stream and updates the current leaf ($stok) and the current token ($ttok)
— defun npNext —

(defun npNext ()
  (declare (special $ inputStream))
  (setq $ inputStream (cdr $ inputStream))
  (npFirstTok))

——

defun npComma

[np Tuple p370]
[np QualifiedDefinition p371]

— defun npComma —

(defun npComma ()
  (np Tuple #' np QualifiedDefinition))

——

defun npTuple

[np ListofFun p445]
[np Comma BackSet p370]
[pf Tuple ListOf p522]

— defun npTuple —

(defun npTuple (|p|)
  (np ListofFun |p| #' np Comma BackSet #' pf Tuple ListOf))

——

defun npComma BackSet

[np EqKey p369]

— defun np Comma BackSet —
(defun npCommaBackSet ()
  (and
    (npEqKey 'comma)
    (or (npEqKey 'backset) t)))

---

defun npQualifiedDefinition

[npQualified p371]
[npDefinitionOrStatement p372]

---

defun npQualified

[npEqKey p369]
[npDefinition p391]
[npTrap p437]
[npPush p367]
[pfWhere p524]
[npPop1 p368]
[npLetQualified p391]

---
defun npDefinitionOrStatement

[npBackTrack p372]
[npGives p372]
[npDef p412]

— defun npDefinitionOrStatement —

(defun npDefinitionOrStatement ()
 (|npBackTrack| #'|npGives| 'def #'|npDef|))

— defun npBackTrack —

(defun npBackTrack (p1 p2 p3)
 (let (a)
   (setq a (|npState|))
   (when (apply p1 nil)
     (cond
      ((|npEqPeek| p2)
       (|npRestore| a)
       (or (apply p3 nil) (|npTrap|)))
      (t t)))))

— defun npGives —

(defun npGives ()
 (|npBackTrack| #'|npExit| 'gives #'|npLambda|))
defun npLambda

(defun npLambda ()
  (or
    (and
      (npVariable)
      (or (npLambda) (npTrap))
      (npPush (pfLam (npPop2) (npPop1))))
    (and
      (npEqKey 'gives)
      (or (npDefinitionOrStatement) (npTrap)))
    (and
      (npEqKey 'colon)
      (or (npType) (npTrap))
      (npEqKey 'gives)
      (or (npDefinitionOrStatement) (npTrap))
      (npPush (pfReturnTyped (npPop2) (npPop1))))))

——

defun npType

(defun npType ()
  (npMatch)
  (npPop1)
  (npWith)
  (npPush)

  — defun npType —
(defun npType ()
  (and
   (npMatch)
   (let ((a (npPop1)))
     (or
      (npWith a)
      (npPush a)))))

defun npMatch

[upLeftAssoc p431]
[upSuch p374]

  — defun npMatch —

(defun npMatch ()
  (npLeftAssoc '(is isnt) #'npSuch))

defun npSuch

[upLeftAssoc p431]
[upLogical p421]

  — defun npSuch —

(defun npSuch ()
  (npLeftAssoc '(bar) #'npLogical))

defun npWith

[npEqKey p369]
[npState p436]
[npCategoryL p377]
[npTrap p437]
[npEqPeek p376]
[npRestore p376]
(defun npWith (extra)
  (let* (a)
    (and
      (npEqKey 'with)
      (progn
        (setq a (npState))
        (or (npCategoryL) (npTrap))
        (if (npEqPeek 'in)
          (progn
            (npRestore a)
            (and
              (or (npVariable) (npTrap))
              (npCompMissing 'in)
              (or (npCategoryL) (npTrap))
              (npPush (pfWith (npPop2) (npPop1) extra))
              (npPush (pfWith (pfNothing) (npPop1) extra)))))))))

(defun npCompMissing (s)
  (or (npEqKey s) (npMissing s)))

defun npMissing

(defun npMissing (s) (npMissing s))

---

(defun npCompMissing (s)
  (or (npEqKey s) (npMissing s)))

defun npMissing

(defun npMissing (s) (npMissing s))
defun npMissing

(defun npMissing (s)
  (declare (special $stok))
  (ncSoftError (tokPosn $stok) "Possibly missing a %1" (list (pname s)))
  (throw 'trappoint 'trapped))

---

defun npRestore

(defun npRestore (x)
  (declare (special $stack $inputStream))
  (setq $inputStream (car x))
  (npFirstTok)
  (setq $stack (cdr x))
  t)

---

defun Peek for keyword s, no advance of token stream

(defun npEqPeek 0 (s)
  (declare (special $ttok $stok))
  (and (eq (caar $stok) '|key|) (eq s $ttok)))

---
defun npCategoryL

(defun |npCategoryL| ()
  (and
   (|npCategory|)
   (|npPush| (|pfUnSequence| (|npPop1|)))))

---

defun npCategory

(defun |npCategory| ()
  (|npPP| #'|npSCategory|))

---

defun npSCategory

(defun |npSCategory| ()
  (|npPP| #'|npSCategory|))
defun npSCategory ()
(let (a)
  (cond
    (npWConditional) (npPush (list (npPop1)))
    (npDefaultValue) t
    (t
      (setq a (npState))
      (cond
        (npPrimary)
          (cond
            (npEqPeek :colon) (npRestore a) (npSignature))
          (t (npRestore a) (or
              (and (npApplication) (npPush (list (pfAttribute (npPop1))))))
              (npTrap)))))
    (t nil))))

-----

defun npSignature

[npSigItemlist p378]
[npPush p367]
[pfWDec p523]
[pfNothing p475]
[npPop1 p368]

-----

defun npSigItemlist

[npListing p379]
[npSigItem p380]
[npPush p367]
[pfListof p475]
— defun npSigItemlist —

(defun npSigItemlist ()
  (and
   (npListing 'npSigItem)
   (npPush (pfListOf (pfAppend (pfParts (npPop1)))))
  ))

— defun npListing —

(defun npListing (p)
  (npList p 'comma '#pfListOf))

——

defun Always produces a list, fn is applied to it

(defun npList (f str1 fn)
  (let (a)
    (declare (special $stack))
    (cond
     ((apply f nil)
      (cond
       ((and (npEqKey str1)
(or (npEqKey 'backset) t)
(or (apply f nil) (npTrap)))
(setq a \$stack\)
(setq \$stack\ nil)
(do () ; while .. do nothing
  (not
    (and (npEqKey str1)
      (or (npEqKey 'backset) t)
      (or (apply f nil) (npTrap))))
  nil))
(setq \$stack\ (cons (nreverse \$stack\) a))
(npPush (funcall fn (cons (npPop3) (npPop2) (npPop1))))
(t (npPush (funcall fn (list (npPop1)))))))
(t (npPush (funcall fn nil))))

defun npSigItem

[npTypeVariable p380]
[npSigDecl p381]
[npTrap p437]

— defun npSigItem —

(defun npSigItem ()
  (and (npTypeVariable) (or (npSigDecl) (npTrap)))))

— defun npTypeVariable —

(defun [npTypeVariable] ()
  (or
    (npParenthesized #\[npTypeVariablelist\])
    (npTypeVariable)
    (npSignatureDefnecie) p381
    [npPush p367]
    [pflistOf p475]
    [npPop1 p368]

— defun npTypeVariable —

(defun npTypeVariable ()
  (or
    (npParenthesized #\[npTypeVariablelist\])
    )
(and (|npSignatureDefinee|) (|npPush| (|pfListOf| (list (|npPop1|)))))

---

defun npSignatureDefinee

[npName p428]
[npInfixOperator p384]
[npPrefixColon p385]

---

defun npTypeVariablelist

[npListing p379]
[npSignatureDefinee p381]

---

defun npSigDecl

[npEqKey p369]
[npType p373]
[npTrap p437]
[npPush p367]
[pfSpread p469]
[pfParts p479]
[npPop2 p368]
[npPop1 p368]
(defun npSigDecl ()
  (and
   (|npEqKey| 'colon)
   (or (|npType|) (|npTrap|))
   (|npPush| (|pfSpread| (|pfParts| (|npPop2|)) (|npPop1|))))

----

defun npPrimary

[npPrimary1 p388]
[npPrimary2 p382]

— defun npPrimary —

(defun npPrimary ()
  (or (npPrimary1) (npPrimary2)))

----

defun npPrimary2

[npEncAp p406]
[npAtom2 p384]
[npAdd p383]
[pfNothing p475]
[npWith p374]

— defun npPrimary2 —

(defun npPrimary2 ()
  (or
   (|npEncAp| #'|npAtom2|)
   (|npAdd| (|pfNothing|))
   (|npWith| (|pfNothing|))))

----

defun npADD

TPDHERE: Note that there is also an npAdd function

[npType p373]
[npPop1 p368]
(defun npADD ()
  (let (a)
    (and
      (npType)
      (progn
        (setq a (npPop1))
        (or
          (npAdd a)
          (npPush a))))))
(progn
  (|npRestore| a)
  (and
    (or (|npVariable|) (|npTrap|))
    (|npCompMissing| 'in)
    (or (|npDefinitionOrStatement|) (|npTrap|))
    (|npPush| (|pfAdd| (|npPop2|) (|npPop1|) extra))))
  (t
    (|npPush| (|pfAdd| (|pfNothing|) (|npPop1|) extra)))))))

---

defun npAtom2

[upInfixOperator p384]
[upAmpersand p428]
[upPrefixColon p385]
[upFromdom p427]

— defun npAtom2 —

(defun npAtom2 ()
  (and
    (or (|npInfixOperator|) (|npAmpersand|) (|npPrefixColon|))
    (|npFromdom|)))

---

defun npInfixOperator

[upInfixOp p385]
[upState p436]
[upEqKey p369]
[upInfixOp p385]
[upPush p367]
[pfSymb p481]
[upPop1 p368]
[tokPosn p635]
[upRestore p376]
[tokConstruct p633]
[tokPart p635]
[$stok p??]

— defun npInfixOperator —
(defun npInfixOperator ()
  (let ((a b))
    (declare (special $ttok| $stok|))
    (or (npInfixOp)
      (progn
        (setq a (npState))
        (setq b $stok)
        (cond
          ((and (npEqKey |'|) (npInfixOp))
           (npPush (pfSymb (npPop1) (tokPosn b))))
          (t
           (npRestore a)
           (cond
             ((and (npEqKey backquote) (npInfixOp))
              (setq a (npPop1))
              (npPush (tokConstruct |idsy| (tokPart a) (tokPosn a))))
             (t
              (npRestore a)
              nil))))))))

---------

defun npInfixOp

[npPushId p433]
[$ttok p??]
[$stok p??]

— defun npInfixOp —

(defun npInfixOp ()
  (declare (special $ttok| $stok|))
  (and
    (eq (caar $stok|) |key|)
    (get $ttok| infgeneric)
    (npPushId))))

---------

defun npPrefixColon

[npEqPeek p376]
[npPush p367]
[tokConstruct p633]
[tokPosn p635]
defun npPrefixColon

(defun npPrefixColon ()
  (declare (special $stok))
  (and
   (npEqPeek 'colon)
   (progn
    (npPush (tokConstruct 'id ':' (tokPosn $stok))
    (npNext))))

defun npApplication

(defun npApplication ()
  (and
   (npDotted (npPrimary))
   (or
    (and
     (npApplication2)
     (npPush (pfApplication (npPop2) (npPop1))))
    t)))

defun npDotted

defun npDotted
(and (apply f nil) (npAnyNo| #'|npSelector|)))

defun npAnyNo

fn must transform the head of the stack

— defun npAnyNo 0 —

(defun npAnyNo (fn)
  (do () ((not (apply fn nil)))) ; while apply do...
t)

defun npSelector

[npEqKey p369]
[npPrimary p382]
[npTrap p437]
[npPush p367]
[pfApplication p483]
[npPop2 p368]
[npPop1 p368]

— defun npSelector —

(defun npSelector ()
  (and
   (npEqKey 'dot)
   (or (npPrimary) (npTrap))
   (npPush (pfApplication (npPop2) (npPop1)))))

defun npApplication2

[npDotted p386]
[npPrimary1 p388]
[npApplication2 p387]
[npPush p367]
[pfApplication p483]
CHAPTER 13. INPUT STREAM PARSER

--- defun npApplication2 ---

(defun npApplication2 ()
(and
 (npDotted) #'npPrimary1)
(or
 (and
 (npApplication2)
 (npPush (pfApplication (npPop2) (npPop1)))))
 t))

--- defun npPrimary1 ---

(defun npPrimary1 ()
(or
 (npEncAp #'npAtom1)
 (npLet)
 (npFix)
 (npMacro)
 (npBPileDefinition)
 (npDefn)
 (npRule)))

--- defun npMacro ---

(defun npMacro ()
(npPP)
(npMdef p389)
— defun npMacro —

(defun |npMacro| ()
  (and
   (|npEqKey| 'macro)
   (|npPP| #'|npMdef|)))

defun npMdef

TPDHERE: Beware that this function occurs with uppercase also

[npQuiver p422]
[pfCheckMacroOut p470]
[npPop1 p368]
[npDefTail p419]
[npTrap p437]
[npPop1 p368]
[npPush p367]
[pfMacro p507]
[pfPushMacroBody p473]

— defun npMdef —

(defun |npMdef| ()
  (let (body arg op tmp)
    (when (|npQuiver|) ; [op, arg] := pfCheckMacroOut(npPop1())
      (setq tmp (|pfCheckMacroOut| (|npPop1|)))
      (setq op (car tmp))
      (setq arg (cadr tmp))
      (or (|npDefTail|) (|npTrap|))
      (setq body (|npPop1|))
      (if (null arg)
       (|npPush| (|pfMacro| op body))
       (|npPush| (|pfMacro| op (|pfPushMacroBody| arg body))))))

defun npMDEF

TPDHERE: Beware that this function occurs with lowercase also

[npBackTrack p372]
[npStatement p395]
--- defun npMDEF ---

(defun npMDEF ()
  (npBackTrack #'npStatement 'mdef #'npMDEFinition))

---

defun npMDEFinition

(defun npMDEFinition ()
  (npPP #'npMdef))

---

defun npFix

(defun npFix ()
  (and
   (npEqKey 'fix)
   (npPP #'npDef)
   (npPush (pfFix (npPop1)))))

---

defun npLet

(defun npLet ()
  (and
   (npEqKey 'let)
   (npPP #'npDefinitionOrStatement)
   (npPush (if (npLetQualified) (npPop1)))))

---
— defun npLet —

(defun npLet ()
  (npLetQualified #'npDefinitionOrStatement)))

——

defun npLetQualified

[npEqKey p369]
[npDefinition p391]
[npTrap p437]
[npCompMissing p375]
[npPush p367]
[pfWhere p524]
[npPop2 p368]
[npPop1 p368]

— defun npLetQualified —

(defun npLetQualified (f)
  (and
   (npEqKey 'let)
   (or (npDefinition) (npTrap))
   (npCompMissing 'in)
   (or (funcall f) (npTrap))
   (npPush (pfWhere (npPop2) (npPop1))))))

——

defun npDefinition

[npPP p434]
[npDefinitionItem p392]
[npPush p367]
[pfSequenceToList p468]
[npPop1 p368]

— defun npDefinition —

(defun npDefinition ()
  (and
   (npPP #\npDefinitionItem))
CHAPTER 13. INPUT STREAM PARSER

(defun npDefinitionItem ()
  (let (a)
    (or (npTyping)
        (npImport)
        (progn
          (setq a (npState))
          (cond
            ((npStatement)
              (cond
                ((npEqPeek) 'def)
                (npRestore a)
                (npDef))
              (t
                (npRestore a)
                (or (npMacro) (npDefn))))
            (t (npTrap)))))))

--- defun npDefinitionItem ---

defun npTyping

(defun npTyping ()
  (let (a)
    (or (npEqKey)
        (npDefaultItemList)
        (npTrap)
        (npPush)))

--- defun npTyping ---

--- defun npTyping ---

(defun npTyping ()
   (and
    (npEqKey 'default)
    (or (npDefaultItemlist) (npTrap))
    (npPush (npTyping (npPop1)))))

---

defun npDefaultItemlist

(defun npDefaultItemlist ()
   (and
    (npPC #'npSDefaultItem)
    (npPush (pfUnSequence (npPop1)))))

---

defun npSDefaultItem

(defun npSDefaultItem ()
   (and
    (npListing #'npDefaultItem)
    (npPush (pfAppend (pfParts (npPop1))))))
CHAPTER 13. INPUT STREAM PARSER

(defun npDefaultItem ()
  (and
   (npTypeVariable)
   (or (npDefaultDecl) (npTrap))))

(defun npDefaultDecl ()
  (and
   (npEqKey 'colon)
   (or (npType) (npTrap)))
   (npPush (pfSpread (pfParts)) (npPop2)) (npPop1))))

(defun npDefaultDecl ()
  (and
   (npEqKey 'colon)
   (or (npType) (npTrap))
   (npPush (npPop2) (npPop1)))
   (npPush (npPop1)))

(defun npDefaultDecl ()
  (and
   (npEqKey 'colon)
   (or (npType) (npTrap))
   (npPush (npPop2) (npPop1)))
   (npPush (npPop1)))
defun npStatement

[npExpress p403]
[npLoop p399]
[npIterate p399]
[npReturn p402]
[npBreak p399]
[npFree p398]
[npImport p405]
[npInline p398]
[npLocal p397]
[npExport p395]
[npTyping p392]
[npVoid p403]

— defun npStatement —

(defun |npStatement| ()
(or
  (|npExpress|)
  (|npLoop|)
  (|npIterate|)
  (|npReturn|)
  (|npBreak|)
  (|npFree|)
  (|npImport|)
  (|npInline|)
  (|npLocal|)
  (|npExport|)
  (|npTyping|)
  (|npVoid|))

———

defun npExport

[npEqKey p369]
[npLocalItemlist p396]
[npTrap p437]
[npPush p367]
[pfExport p494]
[npPop1 p368]

— defun npExport —

(defun |npExport| ()
(and
  (npEqKey 'export)
  (or (npLocalItemlist) (npTrap))
  (npPush (pfExport (npPop1))))

defun npLocalItemlist

(defun npLocalItemlist ()
  (and
   (npPC #'npSLocalItem)
   (npPush (pfUnSequence (npPop1))))

— defun npLocalItemlist —

defun npSLocalItem

(defun npSLocalItem ()
  (and
   (npListing #'npLocalItem)
   (npPush (pfAppend (pfParts (npPop1))))

— defun npSLocalItem —
defun npLocalItem

[npTypeVariable p380]
[npLocalDecl p397]

— defun npLocalItem —

(defun |npLocalItem| ()
(and
  (|npTypeVariable|)
  (|npLocalDecl|)))

defun npLocalDecl

[npEqKey p309]
[npType p373]
[npTrap p437]
[npPush p367]
[pfSpread p469]
[pfParts p479]
[npPop2 p368]
[npPop1 p368]
[pfNothing p475]

— defun npLocalDecl —

(defun |npLocalDecl| ()
  (or
    (and
      (|npEqKey| 'colon)
      (or (|npType|) (|npTrap|))
      (|npPush| (|pfSpread| (|pfParts| (|npPop1|)) (|npPop1|)))
      (|npPush| (|pfSpread| (|pfParts| (|npPop1|)) (|pfNothing|)))))

defun npLocal

[npEqKey p309]
[npLocalItemlist p396]
[npTrap p437]
[npPush p367]
--- defun npLocal ---

(defun npLocal ()
  (and
   (npEqKey 'local)
   (or (npLocalItemlist) (npTrap))
   (npPush (npLocal (npPop1)))))

--- defun npFree ---

(defun npFree ()
  (and
   (npEqKey 'free)
   (or (npLocalItemlist) (npTrap))
   (npPush (npFree (npPop1)))))

--- defun npInline ---

(defun npInline ()
  (npAndOr 'inline #'npQualTypelist #'npInline))
defun npIterate

[npEqKey p369]
[npPush p367]
[pfIterate p501]
[pfNothing p475]

— defun npIterate —

(defun |npIterate| ()
  (and (|npEqKey| 'iterate) (|npPush| (|pfIterate| (|pfNothing|))))))

— defun npBreak —

(defun |npBreak| ()
  (and (|npEqKey| 'break) (|npPush| (|pfBreak| (|pfNothing|))))))

— defun npLoop —

(defun |npLoop| ()

(or
  (and
    (npIterators)
    (npCompMissing 'repeat)
    (or (npAssign) (npTrap))
    (npPush (|pfLp| (|npPop2|) (|npPop1|))))
  (and
    (npEqKey 'repeat)
    (or (npAssign) (npTrap))
    (npPush (|pfLoop1| (|npPop1|)))))

defun npIterators
[|npForIn| p402]
[|npZeroOrMore| p401]
[|npIterator| p400]
[|npPush| p367]
[|npPop2| p368]
[|npPop1| p368]
[|npWhile| p402]
[|npIterators| p400]

  — defun npIterators —

(defun |npIterators| ()
  (or
    (and
      (|npForIn|)
      (|npZeroOrMore| #'|npIterator|)
      (|npPush| (cons (|npPop2|) (|npPop1|))))
    (and
      (|npWhile|)
      (or
        (and (|npIterators|) (|npPush| (cons (|npPop2|) (|npPop1|))))
        (|npPush| (list (|npPop1|)))))))

  ———

defun npIterator
[|npForIn| p402]
[|npSuchThat| p401]
[|npWhile| p402]
— defun npIterator —

(defun npIterator ()
(or
 (npForIn)
 (npSuchThat)
 (npWhile)))

— defun npSuchThat —

(defun npSuchThat ()
(npAndOr 'bar #'npLogical #'npSuchthat))

— defun Apply argument 0 or more times —

(defun npZeroOrMore (f)
(let (a)
 (declare (special $stack))
 (cond
   ((apply f nil)
    (setq a $stack)
    (setq $stack nil)
    (do () ((not (apply f nil))) ; while .. do
    (setq $stack (cons (nreverse $stack) a))
   (npPush (cons (npPop2) (npPop1))))
   (t (progn (npPush nil) t))))

[490x686]401
defun npWhile

[npAndOr p406]
[npLogical p421]
[pfWhile p525]

— defun npWhile —

(defun npWhile ()
  (npAndOr 'while #'npLogical #'pfWhile))

defun npForIn

[npEqKey p369]
[npVariable p437]
[npTrap p437]
[npCompMissing p375]
[npBy p424]
[npPush p367]
[pfForin p496]
[npPop2 p368]
[npPop1 p368]

— defun npForIn —

(defun npForIn ()
  (and
   (npEqKey 'for)
   (or (npVariable) (npTrap))
   (npCompMissing 'in)
   (or (npBy) (npTrap))
   (npPush (pfForin (npPop2) (npPop1)))))

defun npReturn

[npEqKey p369]
[npExpress p403]
--- defun npReturn ---

(defun npReturn ()
  (and
   (npEqKey 'return)
   (or
    (npExpress)
    (npPush (pfNothing))))
  (or
   (and
    (npEqKey 'from)
    (or (npName) (npTrap))
    (npPush (pfReturn (npPop2) (npPop1)))
    (npPush (pfReturnNoName (npPop1)))))

---

defun npVoid

(defun npVoid ()
  (npAndOr 'do #'npStatement #'pfNovalue))

---

defun npExpress

(defun npExpress ()
  (npExpress1)
  (npIterators))
— defun npExpress —

(defun npExpress ()
  (and
   (|npExpress1|)
   (or
    (and
     (|npIterators|)
     (|npPush| (|pfCollect| (|npPop2|) (|pfListOf| (|npPop1|)))))
   t)))

— defun npExpress1 —

(defun npExpress1 ()
  (or (|npConditionalStatement|) (|npADD|)))

— defun npConditionalStatement —

(defun npConditionalStatement ()
  (|npConditional| #'|npQualifiedDefinition|))
defun npImport

[npAndOr p406]
[npQualTypelist p405]
[pfImport p501]

--- defun npImport ---

(defun |npImport| ()
  (|npAndOr| 'import #'|npQualTypelist| #'|pfImport|))

---

defun npQualTypelist

[npPC p??]
[npSQualTypelist p405]
[npPush p367]
[pfUnSequence p523]
[npPop1 p368]

--- defun npQualTypelist ---

(defun |npQualTypelist| ()
  (and
   (|npPC| #'|npSQualTypelist|)
   (|npPush| (|pfUnSequence| (|npPop1|)))))

---

defun npSQualTypelist

[npListing p379]
[npQualType p406]
[npPush p367]
[pfParts p479]
[npPop1 p368]

--- defun npSQualTypelist ---

(defun |npSQualTypelist| ()
  (and
   (|npListing| #'|npQualType|)
   (|npPush| (|pfParts| (|npPop1|)))))
defun npQualType

[npType p373]
[npPush p367]
[pfQualType p512]
[npPop1 p368]
[pfNothing p475]

(defun npQualType ()
  (and
   (npType)
   (npPush (pfQualType (npPop1) (pfNothing)))))

---

defun npAndOr

[npEqKey p369]
[npTrap p437]
[npPush p367]
[npPop1 p368]

(defun npAndOr (keyword p f)
  (and
   (npEqKey keyword)
   (or (apply p nil) (npTrap))
   (npPush (funcall f (npPop1)))))

---

defun npEncAp

[npAnyNo p387]
[npEncl p407]
[npFromdom p427]

(defun npEncAp ()
  (and
   (npAnyNo)
   (npEncl)
   (npFromdom)))
(defun npEncAp (f)
  (and (apply f nil) (#'npEnccl) (npFromdom))))

---

defun npEncl

[npBDefinition p409]
[npPush p367]
[pfApplication p483]
[npPop2 p368]
[npPop1 p368]

---

(defun npEncl ()
  (and
    (#'npBDefinition)
    (npPush (pfApplication (npPop2) (npPop1)))))

---

defun npAtom1

[npPDefinition p408]
[npName p428]
[npConstTok p408]
[npDollar p408]
[npBDefinition p409]
[npFromdom p427]

---

(defun npAtom1 ()
  (or
    (#'npPDefinition)
    (and
      (or (#'npName) (#'npConstTok) (#'npDollar) (#'npBDefinition)
        (#'npFromdom)))))

---


\begin{figure}[ht]
\centering
\includegraphics[width=\textwidth]{image}
\caption{Example figure caption.}
\end{figure}

\begin{table}[ht]
\centering
\begin{tabular}{|l|l|}
\hline
Header 1 & Header 2 \\
\hline
Row 1 & Column 1 \\
Row 2 & Column 2 \\
\hline
\end{tabular}
\caption{Example table caption.}
\end{table}

\begin{equation}
\sum_{i=1}^{n} a_i = b
\end{equation}

---

\section{Section Title}

\begin{itemize}
\item Item 1
\item Item 2
\end{itemize}

---

\begin{itemize}
\setlength\itemsep{0em}
\item Item 1
\item Item 2
\end{itemize}

---

\section{Section Title}

\begin{enumerate}
\item Item 1
\item Item 2
\end{enumerate}

---

\begin{enumerate}
\setlength\itemsep{0em}
\item Item 1
\item Item 2
\end{enumerate}
— defun npConstTok —

(defun npConstTok ()
  (let ((b a))
    (declare (special $stok))
    (cond
      ((member (tokType $stok) ’(integer string char float command))
       (npPush $stok))
      (npNext)))
    (((npEqPeek ’‘))
     (setq a $stok)
     (setq b (npState))
     (npNext))
    (cond
      ((and (npPrimary1))
       (npPush (pfSymb (npPop1) (tokPosn a)))
       t)
      (t (npRestore b) nil)))
    (t nil)))

——

defun npBDefinition

(defun npBDefinition ()
  (or
    (npPDefinition)
    (npBracketed ’#’npDefinitionlist))))

——
defun npBracketed

[ npParened p410 ]
[ npBracked p410 ]
[ npBraced p411 ]
[ npAngleBared p411 ]

— defun npBracketed —

(defun npBracketed (f)
  (or
   (npParened f)
   (npBracked f)
   (npBraced f)
   (npAngleBared f)))

——

defun npParened

[ npEnclosed p436 ]
[ pfParen p511 ]

— defun npParened —

(defun npParened (f)
  (or
   (npEnclosed '('|( ')| #'|pfParen| f)
   (npEnclosed '('|\| '\|)| #'|pfParen| f)))

——

defun npBracked

[ npEnclosed p436 ]
[ pfBracket p487 ]
[ pfBracketBar p487 ]

— defun npBracked —

(defun npBracked (f)
  (or
   (npEnclosed '][' #'|pfBracket| f)
   (npEnclosed '!]\| '\]| #'|pfBracketBar| f)))

——
defun npBraced

(defun npBraced (f)
  (or (npEnclosed '{' #'pfBrace f)
      (npEnclosed '|' '{' '|' '}| #'pfBraceBar f)))

defun npAngleBared

(defun npAngleBared (f)
  (npEnclosed '|<|'| '>| #'pfHide f))

defun npDefn

(defun npDefn ()
  (and
   (npEqKey 'defn)
   (npPP #'npDef))))
defun npDef

[npMatch p374]
[pfCheckItOut p469]
[npPop1 p368]
[npDefTail p419]
[npTrap p437]
[npPop1 p368]
[npPush p367]
[pfDefinition p491]
[pfPushBody p479]

— defun npDef —

(defun npDef ()
 (let (body rt arg op tmp1)
 (when (npMatch)
 ; [op, arg, rt] := pfCheckItOut(npPop1())
 (setq tmp1 (pfCheckItOut (npPop1))
 (setq op (car tmp1))
 (setq arg (cadr tmp1))
 (setq rt (caddr tmp1))
 (or (npDefTail) (npTrap))
 (setq body (npPop1))
 (if (null arg)
 (npPush (pfDefinition op body))
 (npPush (pfDefinition op (pfPushBody rt arg body)))))))

——

defun npBPileDefinition

[npPileBracketed p413]
[npPileDefinitionlist p413]
[npPush p367]
[pfSequence p517]
[pfListOf p475]
[npPop1 p368]

— defun npBPileDefinition —

(defun npBPileDefinition ()
 (and
 (npPileBracketed #'|npPileDefinitionlist|)
 (npPush (pfSequence (pfListOf (npPop1))))))
defun npPileBracketed

(defun |npPileBracketed| (f)
  (cond
    ((|npEqKey| 'settab)
      (cond
        ((|npEqKey| 'backtab)
         (|npPush| (|pfNothing|))) ; never happens
        ((and (apply f nil)
             (or (|npEqKey| 'backtab) (|npMissing| '|backtab|)))
         (|npPush| (|pfPile| (|npPop1|))))
      (t nil))
    (t nil)))

———

defun npPileDefinitionlist

(defun |npPileDefinitionlist| ()
  (and
    (|npListAndRecover| #'|npDefinitionlist|)
    (|npPush| (|pfAppend| (|npPop1|))))

———
defun npListAndRecover

[trappoint p??]
[npRecoverTrap p415]
[syGeneralErrorHere p416]
[npEqKey p369]
[npEqPeek p376]
[npNext p369]
[npPop1 p368]
[npPush p367]
[$inputStream p??]
[$stack p??]

— defun npListAndRecover —

(defun npListAndRecover (f)
  (let (found c done b savestack)
    (declare (special $inputStream $stack))
    (setq savestack $stack)
    (setq $stack nil)
    (setq c $inputStream)
    (do ()
      (done)
      (setq found (catch 'trappoint (apply f nil)))
      (cond
       ((eq found 'trapped)
        (setq $inputStream c)
        (npRecoverTrap))
       ((null found)
        (setq $inputStream c)
        (syGeneralErrorHere) (npRecoverTrap)))
      (cond
       (npEqKey 'backset) (setq c $inputStream))
       (npEqPeek 'backtab) (setq done t)
       (t
        (setq $inputStream c)
        (syGeneralErrorHere)
        (npRecoverTrap))
      (cond
       (npEqPeek 'backtab) (setq done t)
       (t
        (npNext)
        (setq c $inputStream)))))))
    (setq b (cons (npPop1) b))
    (setq $stack savestack)
    (npPush (nreverse b))))

——
defun npRecoverTrap

[npFirstTok p367]
[tokPosn p635]
[npMoveTo p415]
[syIgnoredFromTo p416]
[npPush p367]
[pfWrong p526]
[pfDocument p476]
[pfListOf p475]
[$stok p??]

(defun npRecoverTrap ()
(let (pos2 pos1)
(declare (special $stok))
(npFirstTok)
(setq pos1 (tokPosn $stok))
(npMoveTo 0)
(setq pos2 (tokPosn $stok))
(syIgnoredFromTo pos1 pos2)
(npPush)
(list (pfWrong ((pfDocument (list "pile syntax error"))
(pfListOf nil)))))

(defun npMoveTo (n)
(let (pos2 pos1)
(declare (special $inputStream))
(cond ((null $inputStream) t)
((eq pos1 0) t)
(t (npMoveTo (- n))))))
((npEqPeek 'backset)
 (cond
  ((eql |n| 0) t)
  (t (npNext|) (npMoveTo| |n|))))
((npEqKey 'settab) (npMoveTo| (+ |n| 1)))
(t (npNext|) (npMoveTo| |n|))))

---

defun syIgnoredFromTo

[pfGlobalLinePosn p465]
[ncSoftError p574]
[FromTo p602]
[From p602]
[To p602]

—— defun syIgnoredFromTo ——

(defun |syIgnoredFromTo| (pos1 pos2)
 (cond
  ((equal (pfGlobalLinePosn pos1) (pfGlobalLinePosn pos2))
   (ncSoftError| (FromTo pos1 pos2) "Ignored." nil))
  (t
   (ncSoftError| (From pos1) "Ignored from here" nil)
   (ncSoftError| (To pos2) "to here." nil))))

——

defun syGeneralErrorHere

[sySpecificErrorHere p416]

—— defun syGeneralErrorHere ——

(defun |syGeneralErrorHere| ()
  (|sySpecificErrorHere| " Improper syntax." nil))

——

defun sySpecificErrorHere

[sySpecificErrorAtToken p417]
[$stok p??]
— defun sySpecificErrorHere —
(defun |sySpecificErrorHere| (key args)
    (declare (special |$stok|))
    (|sySpecificErrorAtToken| |$stok| key args))

——-
defun sySpecificErrorAtToken
[ncSoftError p574]
[tokPosn p635]

— defun sySpecificErrorAtToken —
(defun |sySpecificErrorAtToken| (tok key args)
    (|ncSoftError| (|tokPosn| tok) key args))

——-
defun npDefinitionlist
[npSemiListing p417]
[npQualDef p369]

— defun npDefinitionlist —
(defun |npDefinitionlist| ()
    (|npSemiListing| #'|npQualDef|))

——-
defun npSemiListing
[npListofFun p445]
[npSemiBackSet p418]
[pfAppend p485]

— defun npSemiListing —
(defun |npSemiListing| (p)
    (|npListofFun| p #'|npSemiBackSet| #'|pfAppend|))
defun npSemiBackSet
[npEqKey p369]
— defun npSemiBackSet —
(defun npSemiBackSet ()
(and (npEqKey 'semicolon) (or (npEqKey 'backset) t)))

defun npRule
[npEqKey p369]
[npPP p434]
[npSingleRule p418]
— defun npRule —
(defun npRule ()
(and
 (npEqKey 'rule)
 (npPP #'npSingleRule))

defun npSingleRule
[npQuiver p422]
[npDefTail p419]
[npTrap p437]
[npPush p367]
[pfRule p515]
[npPop2 p368]
[npPop1 p368]
— defun npSingleRule —
(defun npSingleRule ()
(when (npQuiver)
 (or (npDefTail) (npTrap)))
defun npDefTail

[npEqKey p369]
[npDefinitionOrStatement p372]

— defun npDefTail —

(defun npDefTail ()
(and
 (or (npEqKey 'def) (npEqKey 'mdef))
 (npDefinitionOrStatement))

defun npDefaultValue

[npEqKey p369]
[npDefinitionOrStatement p372]
[npTrap p437]
[npPush p367]
[pfAdd p482]
[pfNothing p475]
[npPop1 p368]

— defun npDefaultValue —

(defun npDefaultValue ()
(and
 (npEqKey 'default)
 (or (npDefinitionOrStatement) (npTrap))
 (npPush (list (pfAdd (pfNothing) (npPop1) (pfNothing))))))

defun npWConditional

[npConditional p420]
[npPush p367]
--- defun npWConditional ---

(defun npWConditional (f)
  (when (npConditional f) (npPush (npTweakIf (npPop1)))))

--- defun npConditional ---

(defun npConditional (f)
  (cond
    ((and (npEqKey 'IF)
      (or (npLogical) (npTrap))
      (or (npEqKey 'backset) t)) nil)
    (t nil))
  (cond
    ((npEqKey 'settab) nil)
    (cond
      ((npEqKey 'then)
       (and (or (apply f nil) (npTrap))
        (npElse f)
        (t (npMissing 'then)))
      ((npEqKey 'backtab)
       (t (npMissing 'backtab)))
      (t nil))
    )
  )
)

--- defun npElse ---

(defun npElse (f)
  (cond
    ((npEqKey 'then)
     (and (or (apply f nil) (npTrap)) (npElse f))
     (t (npMissing 'then)))
    (t nil))
)
— defun npElse —

(defun npElse (f)
  (let (a)
    (setq a (npState))
    (cond
      ((npBacksetElse)
       (and
        (or (apply f nil) (npTrap))
        (npPush (pIf ((npPop3) (npPop2) (npPop1))))))
      (t
       (npRestore a)
       (npPush (pIfThenOnly (npPop2) (npPop1))))))

— defun npBacksetElse —

TPDHERE: Well this makes no sense.

— defun npBacksetElse —

(defun npBacksetElse ()
  (if (npEqKey 'backset)
    (npEqKey 'else)
    (npEqKey 'else)))

— defun npLogical —

TPDHERE: Well this makes no sense.

— defun npLogical —
(defun npLogical ()
  (|npLeftAssoc| '(or) #'|npDisjand|))

---

defun npDisjand

[npLeftAssoc p431]
[npDiscrim p422]

---

defun npDiscrim

[npLeftAssoc p431]
[npQuiver p422]

---

defun npQuiver

[npRightAssoc p430]
[npRelation p423]
defun npRelation

[npLeftAssoc p431]
[npSynthetic p423]

— defun npRelation —

(defun npRelation ()
  (npLeftAssoc '(equal notequal lt le gt ge oangle cangle) #'npSynthetic))

— undef npSynthetic —

(defun npSynthetic ()
  (cond
    (npBy ((lambda ()
              (loop
                (cond
                  ((not (and (npAmpersandFrom)
                    (or (npBy)
                      (progn
                        (npPush (npApplication (npPop2) (npPop1)))
                        nil)))))
                (return nil))
                t
                (npPush (npInfApplication (npPop2) (npPop2) (npPop1))))
                t)
        t nil)))
defun npBy
  (defun npBy ()
    (npLeftAssoc '(by) #'|npInterval|))

defun npInterval
  (defun npInterval ()
    (and
      (npArith)
      (or
        (and
          (npSegment))
        (or
          (and
            (npEqPeek 'bar)
            (npPush ((pfApplication (npPop1) (npPop1) (npPop1))))
          (and
            (npArith)
            (npPush ((pfInfApplication (npPop2) (npPop2) (npPop1) (npPop1))))
          (npPush ((pfApplication (npPop1) (npPop1))))
          t)))))
defun npSegment

[|npEqPeek| p376]
[|npPushId| p433]
[|npFromdom| p427]

— defun npSegment —

(defun |npSegment| ()
  (and (|npEqPeek| 'seg) (|npPushId|) (|npFromdom|)))

------

defun npArith

[|npLeftAssoc| p431]
[|npSum| p425]

— defun npArith —

(defun |npArith| ()
  (|npLeftAssoc| '(mod) #'|npSum|))

------

defun npSum

[|npLeftAssoc| p431]
[|npTerm| p425]

— defun npSum —

(defun |npSum| ()
  (|npLeftAssoc| '(plus minus) #'|npTerm|))

------

defun npTerm

[|npInfGeneric| p432]
[|npRemainder| p426]
[|npPush| p367]
--- defun npTerm ---

(defun npTerm ()
  (or
    (and
      (npInfGeneric '(-+))
    (or
      (and (npRemainder) (push (pfApplication (npPop2) (npPop1))))
    (npRemainder))))

---

defun npRemainder

(defun npRemainder ()
  (npLeftAssoc '(rem quo) 'npProduct))

---

defun npProduct

(defun npProduct ()
  (npLeftAssoc
    '(* / \ /\ /\ )
   'npPower))
defun npPower
[ npRightAssoc p430 ]
[ npColon p442 ]

---
defun npPower ---
(defun \npPower ()
  (\npRightAssoc '(power carat) '#\npColon))

---
defun npAmpersandFrom
[ npAmpersand p428 ]
[ npFromdom p427 ]

---
defun npAmpersandFrom ---
(defun \npAmpersandFrom ()
  (and (\npAmpersand) (\npFromdom)))

---
defun npFromdom
[ npEqKey p369 ]
[ npApplication p386 ]
[ npTrap p437 ]
[ npFromdom1 p428 ]
[ npPop1 p368 ]
[ npPush p367 ]
[ pfFromDom p497 ]

---
defun npFromdom ---
(defun \npFromdom ()
  (or
    (and
      (\npEqKey ' $)
      (or (\npApplication) (\npTrap))
      (\npFromdom1 (\npPop1))
      (\npPush (\npfFromDom (\npPop1) (\npPop1))))
    t))
defun npFromdom1

(defun npFromdom1 (c)
  (or
   (and
    (npEqKey '"
    (or (npApplication) (npTrap))
    (npFromdom1 (npPop1))
    (npPush (pfFromDom (npPop1) c)))
   (npPush c)))

---

defun npAmpersand

(defun npAmpersand ()
  (and
   (npEqKey 'ampersand)
   (or (npName) (npTrap)))))

---

defun npName

(defun npName
  (npId)
  (npSymbolVariable))
— defun npName —

(defun npName ()
  (or (npId) (npSymbolVariable)))

——

defvar $npTokToNames

— initvars —

(defvar $npTokToNames (list '~ '# '[] '} {} '}' '}' '}' '}' ''))

——

defun npId

[npPush p367]
[npNext p369]
[tokConstruct p633]
[tokPosn p635]
[$npTokToNames p429]
[ttok p??]
[stok p??]

— defun npId —

(defun npId ()
  (declare (special $npTokToNames $ttok $stok))
  (cond
    ((eq (caar $stok) #'id)
      (npPush $stok)
      (npNext))
    ((and (eq (caar $stok) #'key) (member $ttok $npTokToNames))
      (npPush (list tokConstruct #'id $ttok (list tokPosn $stok)))
      (npNext))
    (t nil)))
defun npSymbolVariable

(defun npSymbolVariable ()
  (let (a)
    (setq a (npState))
    (cond
      ((and (npEqKey 'backquote) (npId))
        (setq a (npPop1))
        (npPush (tokConstruct 'idsy (tokPart a) (tokPosn a))))
      (t (npRestore a) nil))))

— defun npSymbolVariable —

defun npRightAssoc

(defun npRightAssoc (o p)
  (let (a)
    (setq a (npState))
    (cond
      ((apply p nil)
        ((lambda ()
           (loop
            (npPop2 (npPop1 a))))))
      (t (npRestore a) nil))))

— defun npRightAssoc —
(cond
  ((not
       (and
        (npInfGeneric o)
        (or
         (npRightAssoc o p)
         (progn (npPush (pfApplication (npPop2) (npPop1))) nil)))
       (return nil))
    (t
     (npPush (pfInfApplication (npPop2) (npPop2) (npPop1)))
    t)
    (npRestore a)
    nil)))

(defun p o p o p o p = (((p o p) o p) o p)
  p o p o p o p = (((p o p) o p) o p)
  p o p o = (p o p)
  ;npLeftAssoc(operations, parser) =
  ;  if APPLY(parser, nil)
  ;    then
  ;      while npInfGeneric(operations)
  ;        and (APPLY(parser, nil) or
  ;          (npPush pfApplication(npPop2(), npPop1()); false))
  ;        repeat
  ;        npPush pfInfApplication(npPop2(), npPop2(), npPop1())
  ;      true
  ;    else false

[npInfGeneric p432]
[npPush p367]
[pfApplication p483]
[npPop2 p368]
[npPop1 p368]
[pfInfApplication p501]

--- defun npLeftAssoc ---

(defun |npLeftAssoc| (operations parser)
  (when (apply parser nil)
    ((lambda nil
       (loop
        (cond
          (not
(and
  (|npInfGeneric| operations)
  (or
   (apply parser nil)
   (progn (|npPush| (|pfApplication| (|npPop2|) (|npPop1||)) nil)))
  (return nil))
  (t
   (|npPush| (|pfInfApplication| (|npPop2|) (|npPop2|) (|npPop1||)))))))

defun npInfGeneric

[|npDDInfKey p432|]
[|npEqKey p369|]

— defun npInfGeneric —

(defun |npInfGeneric| (s)
  (and
   (|npDDInfKey| s)
   (or (|npEqKey| 'backset) t)))

defun npDDInfKey

[|npInfKey p433|]
[|npState p436|]
[|npEqKey p369|]
[|npPush p367|]
[|pfSymb p481|]
[|npPop1 p368|]
[|tokPosn p635|]
[|npRestore p376|]
[|tokConstruct p633|]
[|tokPart p635|]
[|$stok p??|]

— defun npDDInfKey —

(defun |npDDInfKey| (s)
  (let (b a)
(declare (special $stok$))
(or
  (npInfKey| s)
  (progn
    (setq a (npState|))
    (setq b $stok$)
    (cond
      ((and (npEqKey| '|') (npInfKey| s))
       (npPush| (pfSymb| (npPop1|) (tokPosn| b))))
      (t
       (npRestore| a)
       (cond
        ((and (npEqKey| 'backquote) (npInfKey| s))
         (setq a (npPop1|))
         (npPush| (tokConstruct| 'idsy| (tokPart| a) (tokPosn| a))))
        (t
         (npRestore| a)
         nil))))))

defun npInfKey

[npPushId p433]
[$stok p??]
[$ttok p??]
     — defun npInfKey —

(defun npInfKey| (s)
  (declare (special $ttok$ |$stok$))
  (and (eq (caar |$stok$) '|key|) (member |$ttok$| s) (npPushId|)))

———

defun npPushId

[tokConstruct p633]
[tokPosn p635]
[npNext p369]
[$stack p??]
[$stok p??]
[$ttok p??]
     — defun npPushId —
(defun npPushId ()
(let (a)
(declare (special $stack $stok $ttok))
(setq a (get $ttok 'infgeneric))
(when a (setq $ttok a))
(setq $stack
  (cons (|tokConstruct| 'id| $ttok| (|tokPosn| $stok|) $stack))
($stack)))

---

defvar npPParg

— initvars —

(defvar *npPParg* nil "rewrite npPP without flets, using global scoping")

---

defun npPP

This was rewritten by NAG to remove flet.

[npParened p410]
[npPpf p435]
[npPileBracketed p413]
[npPPg p435]
[npPush p367]
[pfEnSequence p493]
[npPop1 p368]
[npPParg p434]

— defun npPP —

(defun npPP (f)
(declare (special *npPParg*))
(setq *npPParg* f)
(or
 (|npParened| #'npPpf)
 (and (|npPileBracketed| #'npPPg) (|npPush| (|pfEnSequence| (|npPop1|))))
 (funcall f)))

-------
defun npPPff

[npPop1 p368]
[npPush p367]
[$npPParg p434]

— defun npPPff —

(defun npPPff ()
  (and (funcall *npPParg*) (npPush (list (npPop1)))))

———

defun npPPg

[npListAndRecover p414]
[npPPf p435]
[npPush p367]
[pfAppend p485]
[npPop1 p368]

— defun npPPg —

(defun npPPg ()
  (and (npListAndRecover #'npPPf)
       (npPush (pfAppend (npPop1)))))

———

defun npPPf

[npSemiListing p417]
[npPPff p435]

— defun npPPf —

(defun npPPf ()
  (npSemiListing #'npPPff))

———
defun npEnclosed

[npEqKey p369]
[npPush p367]
[pfTuple p522]
[pfListOf p475]
[npMissingMate p440]
[pfEnSequence p493]
[npPop1 p368]
[$stok p??]

---------

<table>
<thead>
<tr>
<th>defun npEnclosed</th>
</tr>
</thead>
</table>
| defun npEnclosed (open close fn f)
| (let (a)
| (declare (special $stok))
| (setq a $stok)
| (when (npEqKey open)
| (cond
|   ((npEqKey close)
|     (npPush (funcall fn a (pfTuple (pfListOf NIL))))))
|   ((and (apply f nil)
|        (or (npEqKey close)
|            (npMissingMate close a)))
|     (npPush (funcall fn a (pfEnSequence (npPop1))))))
| ('t nil)))]

---------

defun npState

[$stack p??]
[$inputStream p??]

---------

<table>
<thead>
<tr>
<th>defun npState</th>
</tr>
</thead>
</table>
| defun npState ()
| (declare (special $stack |$inputStream|))
| (cons |$inputStream| |$stack|))

---------
defun npTrap

(defun npTrap ()
 (declare (special $stok))
 (ncSoftError (tokPosn $stok) "Improper syntax." nil)
 (throw 'trappoint 'trapped))

defun npTrapForm

(defun npTrapForm (x)
 (let (a)
  (setq a (pfSourceStok x))
  (cond
   ((eq a '|NoToken|)
    (syGeneralErrorHere))
   (t
    (ncSoftError (tokPosn a) "Improper syntax." nil)
    (throw 'trappoint 'trapped))))

defun npVariable

(defun npVariable ()
 (npParenthesized npVariablelist npVariableName))
defun npVariable

(defun npVariable ()
  (or
   (npParenthesized #'npVariableList)
   (and (npVariableName) (npPush (pfListOf (list (npPop1)))))))

defun npVariableList

(defun npVariableList ()
  (npListing #'npVariableName))

defun npVariableName

(defun npVariableName ()
  (and
   (npName)
   (or (npDecl) (npPush (pfTyped (npPop1) (pfNothing)))))))
defun npDecl

(defun npDecl ()
  (and
    (npEqKey 'colon)
    (or (npType) (npTrap))
    (npPush (pfTyped (npPop2) (npPop1)))))

———

defun npParenthesized

(defun npParenthesized (f)
  (or (npParenthesize '|(' '||| f) (npParenthesize '|(| '||| f))))

———

defun npParenthesize

(defun npParenthesize ()
  (let (a)
    (declare (special |$stok|))
    (setq a |$stok|))
    (npEqKey p369)
    (npMissingMate p440)
    (npPush p367)
    ($stok p??]

    — defun npParenthesize —

    (defun npParenthesize (open close f)
      (let (a)
        (declare (special |$stok|))
        (setq a |$stok|))

    — defun npParenthesized —

    (defun npParenthesized (f)
      (or (npParenthesize '|(' '||| f) (npParenthesize '|(| '||| f)))

    — defun npDecl —

    (defun npDecl ()
      (and
        (npEqKey 'colon)
        (or (npType) (npTrap))
        (npPush (pfTyped (npPop2) (npPop1)))))

    — defun npParenthesized —

    (defun npParenthesized (f)
      (or (npParenthesize '|(' '||| f) (npParenthesize '|(| '||| f))))
(cond
  ((|npEqKey| open)
   (cond
    ((and (apply f nil)
       (or (|npEqKey| close)
           (|npMissingMate| close a)))
     t)
    (t (|npMissingMate| close a)))
   t)))

defun npMissingMate

--; ncSoftError p574
tokPosn p635
npMissing p375

— defun npMissingMate —

(defun |npMissingMate| (close open)
  (|ncSoftError| (|tokPosn| open) "Missing mate." nil)
  (|npMissing| close))

——

defun npExit

--; npBackTrack p372
npAssign p441
npPileExit p440

— defun npExit —

(defun |npExit| ()
  (|npBackTrack| #'|npAssign| 'exit #'|npPileExit|))

——

defun npPileExit

--; npAssign p441
npEqKey p369
— defun npPileExit —

(defun |npPileExit| ()
  (and
    (|npAssign|)
    (or (|npEqKey| 'exit) (|npTrap|))
    (or (|npStatement|) (|npTrap|))
    (|npPush| (|pfExit| (|npPop2|) (|npPop1|)))))

________

defun npAssign

[|npBackTrack| p372]
[|npMDEF| p389]
[|npAssignment| p441]

— defun npAssign —

(defun |npAssign| ()
  (|npBackTrack| '#|npMDEF| 'becomes '#|npAssignment|)))

________

defun npAssignment

[|npAssignVariable| p442]
[|npEqKey| p369]
[|npTrap| p437]
[|npGives| p372]
[|npPush| p367]
[|pfAssign| p485]
[|npPop2| p368]
[|npPop1| p368]

— defun npAssignment —

(defun |npAssignment| ()

; Additional text here
(and
 (|npAssignVariable|)
 (or (|npEqKey| 'becomes) (|npTrap|))
 (or (|npGives|) (|npTrap|))
 (|npPush| (|pfAssign| (|npPop2|) (|npPop1|))))

---

defun npAssignVariable

[|npColon p442|]
[|npPush p367|]
[|pfListof p475|]
[|npPop1 p368|]

— defun npAssignVariable —

(defun npAssignVariable ()
 (and (|npColon|) (|npPush| (|pfListof| (list (|npPop1|))))))

---

defun npColon

[|npTypified p443|]
[|npAnyNo p387|]
[|npTagged p442|]

— defun npColon —

(defun npColon ()
 (and (|npTypified|) (|npAnyNo| #'|npTagged|)))

---

defun npTagged

[|npTypedForm1 p443|]
[|pfTagged p518|]

— defun npTagged —
(defun npTagged ()
  (|npTypedForm1| 'colon #'|pfTagged|))

defun npTypedForm1

[npEqKey p369]
[npType p373]
[npTrap p437]
[npPush p367]
[npPop2 p368]
[npPop1 p368]

— defun npTypedForm1 —

(defun npTypedForm1 (sy fn)
  (and
   (|npEqKey| sy)
   (or (|npType|) (|npTrap|))
   (|npPush| (funcall fn (|npPop2|) (|npPop1|)))))

——

defun npTypified

[npApplication p386]
[npAnyNo p387]
[npTypeStyle p443]

— defun npTypified —

(defun npTypified ()
  (and (|npApplication|) (|npAnyNo| #'|npTypeStyle|)))

——

defun npTypeStyle

[npCoerceTo p444]
[npRestrict p445]
[npPretend p444]
defun npTypeStyle

(defun npTypeStyle ()
  (or (npCoerceTo) (npRestrict) (npPretend) (npColonQuery)))

---

defun npPretend

(defun npPretend ()
  (npTypedForm 'pretend #'pfPretend))

---

defun npColonQuery

(defun npColonQuery ()
  (npTypedForm 'atat #'pfRetractTo))

---

defun npCoerceTo

(defun npCoerceTo ()
  (npTypedForm 'coerce #'pfCoerceto))
defun npTypedForm

[npEqKey p369]
[npApplication p386]
[npTrap p437]
[npPush p367]
[npPop2 p368]
[npPop1 p368]

— defun npTypedForm —

(defun npTypedForm (sy fn)
  (and
   (npEqKey sy)
   (or (npApplication) (npTrap))
   (npPush (funcall fn (npPop2) (npPop1)))))

— defun npRestrict —

(defun npRestrict ()
  (npTypedForm 'at #'npRestrict))

— defun npListoffun —

(defun npListoffun

[npTrap p437]
[npPush p367]
[npPop3 p368]
[npPop2 p368]
[npPop1 p368]
[$stack p??]

— defun npListoffun —
CHAPTER 13. INPUT STREAM PARSER

(defun npListofFun |npListofFun| (f h g)
  (let (a)
    (declare (special |$stack|))
    (cond
      ((apply f nil)
       (cond
        ((and (apply h nil) (or (apply f nil) (|npTrap|)))
         (setq a |$stack|)
         (setq |$stack| nil)
         (do ()
            ((not (and (apply h nil)
                        (or (apply f nil) (|npTrap|))))
             (setq |$stack| (cons (nreverse |$stack|) a))
             (|npPush| (funcall g (cons (|npPop3|) (cons (|npPop2|) (|npPop1|))))))))
        (t t))
      (t nil))))

13.1 Functions on interpreter objects

Interpreter objects used to be called triples because they had the structure [value, type, environment]. For many years, the environment was not used, so finally in January, 1990, the structure of objects was changed to be (type . value). This was chosen because it was the structure of objects of type Any. Sometimes the values are wrapped (see the function isWrapped to see what this means physically). Wrapped values are not actual values belonging to their types. An unwrapped value must be evaluated to get an actual value. A wrapped value must be unwrapped before being passed to a library function. Typically, an unwrapped value in the interpreter consists of LISP code, e.g., parts of a function that is being constructed. – RSS 1/14/90

These are the new structure functions.

```
| mode | val |
```

Object representation

defmacro mkObj

— defmacro mkObj —

(defmacro mkObj (val mode)
13.1. FUNCTIONS ON INTERPRETER OBJECTS

`(cons ,mode ,val))

Object representation

defmacro mkObjWrap

[wrap p1193]

— defmacro mkObjWrap —

(defun mkObjWrap (val mode)
  `(cons ,mode (|wrap| ,val)))

— defmacro mkObjCode —

(defun mkObjCode (val mode)
  `(cons 'cons (cons (mkq ,mode) (cons ,val nil))))

— defmacro objSetVal —
(defmacro |objSetVal| (obj val)
  `(rplacd ,obj ,val))

---

defmacro objSetMode

— defmacro objSetMode —

(defmacro |objSetMode| (obj mode)
  `(rplaca ,obj ,mode))

---

defmacro objVal

— defmacro objVal —

(defmacro |objVal| (obj)
  `(cdr ,obj))

---

defmacro objValUnwrap

— defmacro objValUnwrap —

(defmacro |objValUnwrap| (obj)
  `(|unwrap| (cdr ,obj)))

---

defmacro objMode

— defmacro objMode —
13.2 Macro handling

(defun phMacro

TPDHERE: The pform function has a leading percent sign
carrier[p'tree,...] -> carrier[p'tree, p'treePremacro,...]

(ncEltQ p638)
(ncPutQ p638)
(macroExpanded p450)
[pform p??]

— defun phMacro —

(defun phMacro (carrier)
  (let (ptree)
    (setq ptree (ncEltQ carrier '\ptree'))
    (ncPutQ carrier '\ptreePremacro ptree)
    (setq ptree (macroExpanded ptree))
    (ncPutQ carrier '\ptree ptree)
    'ok))

— defun macroExpanded —

(defun macroExpanded (pf)
  (let (|$posActive| |$macActive|)
    (declare (special |$posActive| |$macActive|))
    (macExpand pf)))

— defun macExpand —

(defun macExpand (pf)
  (let (|$posActive| |$macActive|)
    (declare (special |$posActive| |$macActive|))
    (setq |$macActive| nil)
    (setq |$posActive| nil)
    (macExpand pf)))

— defun macExpand —

(defun macExpand (pf)
  (let (|$posActive| |$macActive|)
    (declare (special |$posActive| |$macActive|))
    (setq |$macActive| nil)
    (setq |$posActive| nil)
    (macExpand pf)))
13.2. MACRO HANDLING

— defun macExpand —

(defun macExpand (pf)
  (cond
    ((pfWhere? pf) (macWhere pf))
    ((pfLambda? pf) (macLambda pf))
    ((pfMacro? pf) (macMacro pf))
    ((pfId? pf) (macId pf))
    ((pfApplication? pf) (macApplication pf))
    (t (pfMapParts #'macExpand pf))))

——

(defun macApplication
  (pfMapParts p466)
  (macExpand p450)
  (pfApplicationOp p484)
  (pfMLambda? p508)
  (pf0ApplicationArgs p467)
  (mac0MLambdaApply p452)
  ($pfMacros p324)

— defun macApplication —

(defun macApplication (pf)
  (let (args op)
    (declare (special $pfMacros))
    (setq pf (pfMapParts #'macExpand pf))
    (setq op (pfApplicationOp pf))
    (cond
      ((null (pfMLambda? op)) pf)
      (t
       (setq args (pf0ApplicationArgs pf))
       (mac0MLambdaApply op args pf $pfMacros))))

——
defun mac0MLambdaApply

TPDHERE: The pform function has a leading percent sign. fix this

(defun mac0MLambdaApply (mlambda args opf |$pfMacros|)
  (declare (special |$pfMacros|))
  (let (pos body params)
    (declare (special |$posActive| |$macActive|))
    (setq params (|pf0MLambdaArgs| mlambda))
    (setq body (|pfMLambdaBody| mlambda))
    (cond
      ((not (eql (length args) (length params)))
       (setq pos (|pfSourcePosition| opf))
       (|ncHardError| pos "Expected %1 arguments, but received %2." (list (length params) (length args))))
      (t ((lambda (parms p arrgs a) ; for p in params for a in args repeat
          (loop
            (cond
              ((or (atom parms)
                (progn (setq p (car parms)) nil)
                (atom arrgs)
                (progn (setq a (CAR arrgs)) nil))
                (return nil))
              (t
               (cond
                ((null (|pfId?| p))
                 (setq pos (|pfSourcePosition| opf))
                 (|ncHardError| pos "Macro parameter %1f is not an id." (list (|%pform| p))))
                (t
                 (|mac0Define| (|pfIdSymbol| p) '|mparam| a))))))
          params nil args nil))

defun mac0ExpandBody

(defun mac0ExpandBody (body opf $macActive $posActive)
  (declare (special $macActive $posActive))
  (let (posn pf)
    (cond
      ((member body $macActive)
       (setq pf (cadr $posActive))
       (setq posn (pfSourcePosition pf))
       (mac0InfiniteExpansion posn body $macActive))
      (t
       (setq $macActive (cons body $macActive))
       (setq $posActive (cons opf $posActive))
       (macExpand body))))

defun mac0InfiniteExpansion

TPDHERE: The pform function has a leading percent sign. fix this

(defun mac0InfiniteExpansion (posn body active)
  (let (rnames fname tmp1 blist result)
    (setq blist (cons body active))
    (setq tmp1 (mapcar #'mac0InfiniteExpansion:name blist))
    (setq fname (car tmp1)) ;[fname, :rnames] := [name b for b in blist]
    (setq rnames (cadr tmp1)))

defun mac0InfiniteExpansion, name

Returns [state, body] or NIL. Returns [sy, state] or NIL.

--- defun mac0GetMacro, name ---

(defun mac0GetMacro (b)
  (let (bd tmp1 st tmp2 sy name)
    (declare (special $pfMacros))
    ; for [sy, st, bd] in $pfMacros while not name repeat
    ((lambda (macros tmplist)
13.2. MACRO HANDLING

```lisp
(loop
  (cond
   ((or (atom macros)
        (progn (setq tmplist (car macros)) nil)
       name)
     (return nil))
   (t
    (and (consp tmplist)
     (progn
      (setq sy (car tmplist))
      (setq tmp2 (cdr tmplist))
      (and (consp tmp2)
      (progn
       (setq st (car tmp2))
       (setq tmp1 (cdr tmp2))
       (and (consp tmp1)
       (eq (cdr tmp1) nil)
       (progn
        (setq bd (car tmp1))
        t)))))))
     (progn
      (when (eq st '|mlambda|) (setq bd (|pfMLambdaBody| bd)))
      (when (eq bd body) (setq name (list sy st)))))
    (setq macros (cdr macros)))))
```

---

defun macId

defun macId (pf)
(let (body state got sy)
  (declare (special |$posActive| (!$macActive|)))
  (setq sy (|pfIdSymbol| pf))
  (cond
   ((null (setq got (|mac0Get| sy)))))
  (pf)
)
(t
  (setq state (car got))
  (setq body (cadr got))
  (cond
    ((eq state '|mparam|) body)
    ((eq state '|mlambda|) (|pfCopyWithPos| body (|pfSourcePosition| pf)))
    (t
      (|pfCopyWithPos|
       (|mac0ExpandBody| body pf |$macActive| |$posActive|)
       (|pfSourcePosition| pf))))))

•••

defun mac0Get

[ifcdr p??]
[|SpfMacros p324|

  — defun mac0Get —

(defun |mac0Get| (sy)
  (declare (special |$pfMacros|))
  (ifcdr (assoc sy |$pfMacros|)))

•••

defun macWhere

[macWhere,mac p456]
[|SpfMacros p324|

  — defun macWhere —

(defun |macWhere| (pf)
  (declare (special |$pfMacros|))
  (|macWhere,mac| pf |$pfMacros|)))

•••

defun macWhere,mac

[|pfMapParts p466|
[macExpand p450]
13.2. MACRO HANDLING

[\$pfMacros p324]

— defun macWhere,mac —

(defun macWhere,mac (pf \$pfMacros))
  (declare (special \$pfMacros))
  (\$pfMapParts \#’macExpand pf))

----------

defun macLambda

[macLambda,mac p457]
[\$pfMacros p324]

— defun macLambda —

(defun macLambda pf)
  (declare (special \$pfMacros))
  (macLambda,mac pf \$pfMacros))

----------

defun macLambda,mac

[pfMapParts p466]
[macExpand p450]
[\$pfMacros p324]

— defun macLambda,mac —

(defun macLambda,mac pf \$pfMacros))
  (declare (special \$pfMacros))
  (\$pfMapParts \#’macExpand pf))

----------

defun Add appropriate definition the a Macro pform

This function adds the definition and returns the original Macro pform. TPDHERE: The pform function has a leading percent sign. fix this [pfMacroLhs p507]
[pfMacroRhs p507]
--- defun macMacro ---

(defun |macMacro| (pf)
  (let (sy rhs lhs)
    (setq lhs (|pfMacroLhs| pf))
    (setq rhs (|pfMacroRhs| pf))
    (cond
      ((null (|pfId?| lhs))
       (|ncSoftError| (|pfSourcePosition| lhs)
        "%1 is improper for macro definition. Ignored."
        (list (|%pform| lhs)))
       pf)
      (t
       (setq sy (|pfIdSymbol| lhs))
       (|mac0Define| sy
        (cond
         ((|pfMLambda?| rhs) '|mlambda|)
         (t '|mbody|))
        (|macSubstituteOuter| rhs))
        (cond
         ((|pfNothing?| rhs) pf)
         (t (|pfMacro| lhs (|pfNothing|))))))))

--- defun Add a macro to the global pfMacros list ---

(defun mac0Define 0
  (defun |mac0Define| (sy state body)
    (declare (special |$pfMacros|))
    (setq |$pfMacros| (cons (list sy state body) |$pfMacros|)))
13.2. MACRO HANDLING

defun macSubstituteOuter

(defun |macSubstituteOuter| (pform)
  (|mac0SubstituteOuter| (|macLambdaParameterHandling| nil pform) pform))

defun mac0SubstituteOuter

(defun |mac0SubstituteOuter| (replist pform)
  (let (tmplist)
    (cond
      ((|pfId?| pform) (|macSubstituteId| replist pform))
      ((|pfLeaf?| pform) pform)
      ((|pfLambda?| pform)
        (setq tmplist (|macLambdaParameterHandling| replist pform))
        (dolist (p (|pfParts| pform)) (|mac0SubstituteOuter| tmplist p))
        pform)
      (t
        (dolist (p (|pfParts| pform)) (|mac0SubstituteOuter| replist p))
        pform)))))

defun macLambdaParameterHandling

(defun |macLambdaParameterHandling|)

--- defun macLambdaParameterHandling ---

(defun macLambdaParameterHandling (replist pform)
  (let (parlist symlist result)
    (cond
      (pfLeaf? pform) nil
      (pfLambda? pform) ; remove (identifier . replacement) from assoclist
        (setq parlist (mapcar #'pfTypedId (pf0LambdaArgs pform)))
        (setq symlist (mapcar #'pfIdSymbol parlist))
        (dolist (par symlist)
          (setq replist
              (let ((pr (assoc par replist :test #'equal)))
                (when pr (remove par replist :test #'equal)))))
              replist)
      (pfMLambda? pform) ; construct assoclist (identifier . replacement)
        (setq parlist (pf0MLambdaArgs pform)) ; extract parameter list
        (dolist (par parlist (nreverse result))
          (push (cons (pfIdSymbol par)
                        (list (pfAbSynOp par) (gensym) (pfLeafPosition par))
                  result)))
    (t
      (dolist (p (pfParts pform)
        (pfLambdaParameterHandling replist p))))))

---

defun macSubstituteId

(defun macSubstituteId (replist pform)
  (let (ex)
(setq ex (assoc (|pfIdSymbol| pform) replist :test #'eq))
(cond
  (ex
    (rplaca pform (cadr ex))
    (rplacd pform (cddr ex))
    pform)
  (t pform))))
Chapter 14

Pftrees

14.1 Abstract Syntax Trees Overview

The functions create and examine abstract syntax trees. These are called pforms, for short.

The pform data structure

- Leaves: [hd, tok, pos] where pos is optional
- Trees: [hd, tree, tree, ...]
- hd is either an id or (id . alist)

The leaves are:

\[
\begin{align*}
\text{char} & := \text{\textquote{char expr position}} \\
\text{Document} & := \text{\textquote{Document expr position}} \\
\text{error} & := \text{\textquote{error expr position}} \\
\text{expression} & := \text{\textquote{expression expr position}} \\
\text{float} & := \text{\textquote{float expr position}} \\
\text{id} & := \text{\textquote{id expr position}} \\
\text{idsy} & := \text{\textquote{idsy expr position}} \\
\text{integer} & := \text{\textquote{integer expr position}} \\
\text{string} & := \text{\textquote{string expr position}} \\
\text{symbol} & := \text{\textquote{symbol expr position}}
\end{align*}
\]

The special nodes:

\[
\begin{align*}
\text{ListOf} & := \text{\textquote{listOf items}} \\
\text{Nothing} & := \text{\textquote{nothing}} \\
\text{SemiColon} & := \text{\textquote{SemiColon (Body: Expr)}}
\end{align*}
\]

The expression nodes:
Add := ('Add (Base: [Typed], Addin: Expr))
And := ('And left right)
Application := ('Application (Op: Expr, Arg: Expr))
Assign := ('Assign (LhsItems: [AssLhs], Rhs: Expr))
Attribute := ('Attribute (Expr: Primary))
Break := ('Break (From: ? Id))
Coerceto := ('Coerceto (Expr: Expr, Type: Type))
Collect := ('Collect (Body: Expr, Iterators: [Iterator]))
ComDefinition := ('ComDefinition (Doc: Document, Def: Definition))
DeclPart
Definition := ('Definition (LhsItems: [Typed], Rhs: Expr))
DefinitionSequence := (Args: [DeclPart])
Do := ('Do (Body: Expr))
Document := ('Document strings)
DWhere := ('DWhere (Context: [DeclPart], Expr: [DeclPart]))
EnSequence :=
Export := ('Export (Items: [Typed]))
Forin := ('Forin (Lhs: [AssLhs], Whole: Expr))
Free := ('Free (Items: [Typed]))
Fromdom := ('Fromdom (What: Id, Domain: Type))
Hide := ('hide, arg)
If := ('If (Cond: Expr, Then: Expr, Else: ? Expr))
Import := ('Import (Items: [QualType]))
Inline := ('Inline (Items: [QualType]))
Iterate := ('Iterate (From: ? Id))
Lambda := ('Lambda (Args: [Typed], Rets: ReturnedTyped, Body: Expr))
Literal
Local := ('Local (Items: [Typed]))
Loop := ('Loop (Iterators: [Iterator]))
Macro := ('Macro (Lhs: Id, Rhs: ExprorNot))
MLambda := ('MLambda (Args: [Id], Body: Expr))
Not := ('Not arg)
Novalue := ('Novalue (Expr: Expr))
Or := ('Or left right)
Pretend := ('Pretend (Expr: Expr, Type: Type))
QualType := ('QualType (Type: Type, Qual: ? Type))
Restrict := ('Restrict (Expr: Expr, Type: Type))
Retract := ('RetractTo (Expr: Expr, Type: Type))
ReturnTyped := ('returntypwed (type body))
Rule := ('Rule (lhsitems, rhsitems))
Sequence := ('Sequence (Args: [Expr]))
Suchthat := ('Suchthat (Cond: Expr))
Symb := if leaf then symbol else expression
Tagged := ('Tagged (Tag: Expr, Expr: Expr))
TLambda := ('TLambda (Args: [Typed],
               Rets: ReturnedTyped Type, Body: Expr))
Tuple := ('Tuple (Parts: [Expr]))
Typed := ('Typed (Id: Id, Type: ? Type))
Typing := ('Typing (Items: [Typed]))
Until := ('Until (Cond: Expr) NOT USED
WDeclare := ('WDeclare (Signature: Typed, Doc: ? Document))
Where := ('Where (Context: [DeclPart], Expr: Expr))
While := ('While (Cond: Expr))
With := ('With (Base: [Typed], Within: [WithPart]))
Special cases of expression nodes are:

- **Application.** The Op parameter is one of \texttt{and, or, y, l, \{, \}, \{||\}, \[||\], [||]}
- **DeclPart.** The comment is attached to all signatutres in Typing, Import, Definition, Sequence, DWhere, Macro nodes
- **EnSequence.** This is either a Tuple or Sequence depending on the argument
- **Literal.** One of integer symbol expression one zero char string float of the form ("expression expr position")

### 14.2 Structure handlers

**defun pfGlobalLinePosn**

[pGlobalLinePosn p296]

— defun pfGlobalLinePosn —

(defun |pfGlobalLinePosn| (posn)
  (|poGlobalLinePosn| posn))

———

**defun pfCharPosn**

[pCharPosn p599]

— defun pfCharPosn —

(defun |pfCharPosn| (posn)
  (|poCharPosn| posn))

———

**defun pfLinePosn**

[pLinePosn p583]

— defun pfLinePosn —
(defun |pfLinePosn| (posn)
  (|poLinePosn| posn))

---

(defun |pfFileName| (posn)
  (|poFileName| posn))

---

(defun |pfCopyWithPos| (pform pos)
  (if (|pfLeaf?| pform)
      (|pfLeaf| (|pfAbSynOp| pform) (|tokPart| pform) pos)
      (|pfTree| (|pfAbSynOp| pform)
        (loop for p in (|pfParts| pform)
           collect (|pfCopyWithPos| p pos)))))

---

(defun |pfMapParts|)
  (|pfLeaf?|)
  (|pfParts|)
  (|pfTree|)

---
14.2. STRUCTURE HANDLERS

— defun pfMapParts —

(defun pfMapParts (f pform)
  (let (parts1 parts0)
    (if (pfLeaf? pform)
      pform
      (progn
        (setq parts0 (pfParts pform))
        (setq parts1 (loop for p in parts0 collect (funcall f p)))
        (if (reduce #'(lambda (u v) (and u v)) (mapcar #'eq parts0 parts1))
          pform
          (pfTree (pfAbSynOp pform) parts1))))))

defun pf0ApplicationArgs

— defun pf0ApplicationArgs —

(defun pf0ApplicationArgs (pform)
  (pf0FlattenSyntacticTuple (pfApplicationArg pform)))

— defun pf0FlattenSyntacticTuple —

(defun pf0FlattenSyntacticTuple (pform)
  (if (null (pfTuple? pform))
    (list pform)
    (lambda (arg0 arg1 p)
      (loop
        (cond
          ((or (atom arg1) (progn (setq p (car arg1)) nil))
          ...)})
        ...)))
defun pfSourcePosition

(defun |pfSourcePosition| (form)
  (let (pos)
    (declare (special |$nopos|))
    (cond
      ((|pfLeaf?| form) (|pfLeafPosition| form))
      (t
        (setq pos |$nopos|)
        ((lambda (theparts p) ; for p in parts while poNoPosition? pos repeat
          (loop
            (cond
              ((or (atom theparts)
                   (progn (setq p (car theparts)) nil)
                   (not (|poNoPosition?| pos)))
                (return nil))
              (t (setq pos (|pfSourcePosition| p))))
              (setq theparts (cdr theparts)))
           (|pfParts| form) nil)
        pos))))

defun Convert a Sequence node to a list

(defun | pfSequence? | (form)
  (null (| pfSequence? | form)))
14.2. STRUCTURE HANDLERS

— defun pfSequenceToList —

(defun pfSequenceToList (x)
  (if (pfSequence? x)
      (pfSequenceArgs x)
      (pfListOf (list x))))

defun pfSpread

[pfTyped p520]

— defun pfSpread —

(defun pfSpread (arg1 arg2)
  (mapcar #')(lambda (i) (pfTyped i arg2)) arg1))

defun Deconstruct nodes to lists

[pfTagged? p519]
[pfTaggedExpr p519]
[pfNothing p475]
[pfTaggedTag p519]
[pfId? p476]
[pfListOf p475]
[pfTyped p520]
[pfCollect1? p472]
[pfCollectVariable1 p472]
[pfTuple? p522]
[pf0TupleParts p523]
[pfTaggedToTyped p519]
[pfDefinition? p491]
[pfApplication? p485]
[pfFlattenApp p471]
[pfTaggedToTyped1 p474]
[pfTransformArg p474]
[npTrapForm p437]

— defun pfCheckItOut —
\textbf{CHAPTER 14. PFTREES}

(defun pfCheckItOut (x)
  (let (args op ls form rt result)
    (if (pfTagged? x)
      (setq rt (pfTaggedExpr x))
      (setq rt (pfNothing)))
    (if (pfTagged? x)
      (setq form (pfTaggedTag x))
      (setq form x))
    (cond
      ((pfId? form) (list (pfListOf (list (pfTyped form rt))) nil rt))
      ((pfCollect1? form) (list (pfListOf (list (pfCollectVariable1 form))) nil rt))
      ((pfTuple? form) (list (pfListOf
        (dolist (part (pf0TupleParts form) (nreverse result))
          (push (pfTaggedToTyped part) result)))
        nil rt))
      ((pfDefinition? form) (list (pfListOf (list (pfTyped form (pfNothing)))) nil rt))
      ((pfApplication? form)
        (setq ls (pfFlattenApp form))
        (setq op (pfTaggedToTyped1 (car ls)))
        (setq args
          (dolist (part (cdr ls) (nreverse result))
            (push (pfTransformArg part) result)))
        (list (pfListOf (list op)) args rt))
      (t (npTrapForm form))))

---

defun pfCheckMacroOut

[pfId? p476]
[pfApplication? p485]
[pfFlattenApp p471]
[pfCheckId p471]
[pfCheckArg p471]
[npTrapForm p437]

--- defun pfCheckMacroOut ---

(defun pfCheckMacroOut (form)
  (let (args op ls)
    (cond
      ((pfId? form) (list form nil))
      ((pfApplication? form)
14.2. STRUCTURE HANDLERS

```lisp
(defun pfCheckArg
  (defun |pfCheckArg| (args)
    (let (argl)
      (if (|pfTuple?| args)
        (setq argl (|pf0TupleParts| args))
        (setq argl (list args)))
      (|pfListOf| (mapcar #'|pfCheckId| argl))))

(defun pfCheckId
  (defun |pfCheckId| (form)
    (if (null (|pfId?| form))
      (|npTrapForm| form)
      form))

(defun pfFlattenApp
  (defun |pfFlattenApp| (form))
  (setq ls (|pfFlattenApp| form))
  (setq op (|pfCheckId| (car ls)))
  (setq args (mapcar #'|pfCheckArg| (cdr ls)))
  (list op args)
  (t (|npTrapForm| form)))))
```

---

defun pfCheckArg

[pfTuple? p522]
[pf0TupleParts p523]
[pfListOf p475]
[pfCheckId p471]

---

defun pfCheckId

[pfId? p476]
[npTrapForm p437]

---

defun pfFlattenApp

[pfApplication? p485]
[pfCollect1? p472]
--- defun pfFlattenApp ---

(defun pfFlattenApp (x)
  (cond
    (\(|pfApplication?| x)\)
      (cond
        (\(\(|pfCollect1?| x\) (LIST x))\) (t
          (append (pfFlattenApp (|pfApplicationOp| x))
                   (pfFlattenApp (|pfApplicationArg| x)))))))
    (t (list x))))

---

defun pfCollect1?

(defun pfCollect1? (x)
  (let (a)
    (when (\(|pfApplication?| x)\)
      (setq a (|pfApplicationOp| x))
    (when (\(|pfId?| a\) (eq (\(|pfIdSymbol| a\) 'n)) (\(\|\) (\|\))))))

---

defun pfCollectVariable1

(defun pfCollectVariable1 (x)
  (cond
    (\(\(\|\) (\|\))\) (t (list x))))

---

14.2. STRUCTURE HANDLERS

— defun pfCollectVariable1 —

(defun pfCollectVariable1 (x)
  (let (id var a)
    (setq a (|pfApplicationArg| x))
    (setq var (car (|pf0TupleParts| a)))
    (setq id (|pfTaggedToTyped| var))
    (|pfTyped|
      (|pfSuch| (|pfTypedId| id) (cadr (|pf0TupleParts| a)))
      (|pfTypedType| id))))

— defun pfPushMacroBody —

(defun pfPushMacroBody (args body)
  (if (null args)
    body
    (|pfMLambda| (car args) (|pfPushMacroBody| (cdr args) body))))

— defun pfSourceStok —

(defun pfSourceStok (x)
  (cond
    ((|pfLeaf?| x) x)
    ((null (|pfParts| x)) '|NoToken|)
    (t (|pfSourceStok| (|pfFirst| x))))
defun pfTransformArg

(defun |pfTransformArg| (args)
  (let (arglist result)
    (if (|pfTuple?| args)
      (setq arglist (|pf0TupleParts| args))
      (setq arglist (list args)))
    (|pfListOf|
      (dolist (|i| arglist (nreverse result))
        (push (|pfTaggedToTyped1| |i|) result))))

defun pfTaggedToTyped1

(defun |pfTaggedToTyped1| (arg)
  (cond
    ((|pfCollect1?| arg) (|pfCollectVariable1| arg))
    ((|pfDefinition?| arg) (|pfTyped| arg (|pfNothing|)))
    (t (|pfTaggedToTyped1| arg))))

defun pfSuch
(defun |pfSuch| (x y)  
  (|pfInfApplication| (|pfId| '||) x y))

14.3 Special Nodes

defun Create a Listof node

[pfTree p482]

— defun pfListOf —

(defun |pfListOf| (x)  
  (|pfTree| 'listOf x))

defun pfNothing

[pfTree p482]

— defun pfNothing —

(defun |pfNothing| ()  
  (|pfTree| 'nothing nil))

defun Is this a Nothing node?

[pfAbSynOp? p634]

— defun pfNothing? —

(defun |pfNothing?| (form)  
  (|pfAbSynOp?| form 'nothing)))
14.4 Leaves

defun Create a Document node

(defun pfDocument (strings)
  (pfLeaf 'Document strings))

defun Construct an Id node

(defun pfId (expr)
  (pfLeaf 'id expr))

defun Is this an Id node?

(defun pfId? (form)
  (or (pfAbSynOp? form 'id) (pfAbSynOp? form 'idsy))))

defun Construct an Id leaf node

(defun pfIdPos — defun pfIdPos —
(defun |pfIdPos| (expr pos)
  (|pfLeaf| 'id expr pos))

        
        
        
defun Return the Id part
[tokPart p635]

        — defun pfIdSymbol —

(defun |pfIdSymbol| (form)
  (|tokPart| form))

        
        
        
defun Construct a Leaf node
[tokConstruct p633]
[ifcar p633]
[pfNoPosition p636]

        — defun pfLeaf —

(defun |pfLeaf| (x y &rest z)
  (|tokConstruct| x y (or (ifcar z) (|pfNoPosition|))))

        
        
        
defun Is this a leaf node?
[pfAbSynOp p634]

        — defun pfLeaf? —

(defun |pfLeaf?| (form)
  (member (|pfAbSynOp| form)
    '(id idsy symbol string char float expression integer Document error)))

        
        
        —

defun Return the token position of a leaf node
[tokPosn p635]

— defun pfLeafPosition —

(defun |pfLeafPosition| (form)  
  (|tokPosn| form))

——

defun Return the Leaf Token
[tokPart p635]

— defun pfLeafToken —

(defun |pfLeafToken| (form)  
  (|tokPart| form))

——

defun Is this a Literal node?
[pfAbSynOp p634]

— defun pfLiteral? 0 —

(defun |pfLiteral?| (form)  
  (member (|pfAbSynOp| form)  
    ’(|integer| |symbol| |expression| |one| |zero| |char| |string| |float|)))

——

defun Create a LiteralClass node
[pfAbSynOp p634]

— defun pfLiteralClass —

(defun |pfLiteralClass| (form)  
  (|pfAbSynOp| form))

——
defun Return the LiteralString

defun pfLiteralString —

(defun pfLiteralString (form)
  (tokPart form))

---------

defun Return the parts of a tree node

defun pfParts 0 —

(defun pfParts (form)
  (cdr form))

---------

defun Return the argument unchanged

defun pfPile 0 —

(defun pfPile (part)
  part)

---------

defun pfPushBody

(defun pfPushBody (rt args body)
  (cond
    ((null args) body)
defun An S-expression which people can read.

[pfSexpr,strip p480]

---
defun pfSexpr ---

(defun pfSexpr (pform)
  (pfSexpr,strip pform))

---
defun Create a human readable S-expression

[pfId? p476]
[pfIdSymbol p477]
[pfLiteral? p478]
[pfLiteralString p479]
[pfLeaf? p477]
[tokPart p635]
[pfApplication? p485]
[pfApplicationArg p484]
[pfTuple? p522]
[pf0TupleParts p523]
[pfApplicationOp p484]
[pfSexpr,strip p480]
[pfAbSynOp p634]
[pfParts p479]

---
defun pfSexpr,strip ---

(defun pfSexpr,strip (pform)
  (let (args a result)
    (cond
      (((pfId?| pform) (pfIdSymbol| pform))
        (pfIdSymbol pform))
      (((pfLiteral?| pform) (pfLiteralString| pform))
        (pfLiteralString pform))
      (((pfLeaf?| pform) (tokPart| pform))
        (tokPart pform))
      (((pfApplication?| pform)
        (pfApplicationOp pform))
        (pfApplicationOp pform))
      (((pfApplicationArg| pform)
        (pfApplicationArg pform))
        (pfApplicationArg pform))
      (((pfSexpr,strip| pform)
        (pfSexpr,strip pform))
        (pfSexpr,strip pform))
      (((pfTuple?| pform) (pfTuple? pform))
        (pfTuple? pform))
      (((pf0TupleParts| pform) (pf0TupleParts pform))
        (pf0TupleParts pform))
      (((pfAbSynOp| pform) (pfAbSynOp pform))
        (pfAbSynOp pform))
      (((pfParts| pform) (pfParts pform))
        (pfParts pform))
    )))

CHAPTER 14. PFTREES

(((null (cdr args)) (pfLambda| (car args) rt body))
 (t
  (pfLambda| (car args) (pfNothing|)
    (pfPushBody| rt (cdr args) body))))
(setq a (|pf.ApplicationArg| pform))
(if (|pf.Tuple?| a)
    (setq args (|pf.0TupleParts| a))
    (setq args (list a)))
(dolist (p (cons (|pf.ApplicationOp| pform) args) (nreverse result))
    (push (|pf.Sexpr,strip| p) result))
(t
    (cons (|pf.AbbSynOp| pform)
      (dolist (p (|pf.Parts| pform) (nreverse result))
        (push (|pf.Sexpr,strip| p) result)))))))

defun Construct a Symbol or Expression node

[|pfLeaf?| p477]
[|pf.Symbol| p481]
[tokPart p635]
[ifcar p??]
[|pf.Expression| p494]
[|pf.Sexpr| p480]

    — defun pfSymb —

(defun |pf.Symb| (expr &REST optpos)
  (if (|pf.Leaf?| expr)
      (|pf.Symbol| (|tokPart| expr) (ifcar optpos))
      (|pf.Expression| (|pf.Sexpr| expr) (ifcar optpos)))))

    ——

defun Construct a Symbol leaf node

[|pfLeaf| p477]
[ifcar p??]

    — defun pfSymbol —

(defun |pf.Symbol| (expr &rest optpos)
  (|pf.Leaf| '|symbol| expr (ifcar optpos)))

    ——
defun Is this a Symbol node?

— defun pfSymbol? —

(defun pfSymbol? (form)
  (if pfAbSynOp? form 'symbol))

——

defun Return the Symbol part

— defun pfSymbolSymbol —

(defun pfSymbolSymbol (form)
  (if tokPart form))

——

14.5 Trees

defun Construct a tree node

— defun pfTree 0 —

(defun pfTree (x y)
  (cons x y))

——

defun Construct an Add node

— defun pfAdd —
(defun pfAdd (pfbase pfaddin &rest addon)
  (let (lhs)
    (if addon
      (setq lhs addon)
      (setq lhs (pfNothing)))
    (pfTree 'Add (list pfbase pfaddin lhs))))

defun Construct an And node

(defun pfAnd (pfleft pfright)
  (pfTree 'And (list pfleft pfright)))

defun pfAttribute

(defun pfAttribute (pfexpr)
  (pfTree 'Attribute (list pfexpr)))

defun Return an Application node
defun Return the Arg part of an Application node

    — defun pfApplicationArg 0 —

    (defun |pfApplicationArg| (pf)
      (caddr pf))

———

defun Return the Op part of an Application node

    — defun pfApplicationOp 0 —

    (defun |pfApplicationOp| (pf)
      (cadr pf))

———

defun Is this an And node?

[pfAbSynOp? p634]

    — defun pfAnd? —

    (defun |pfAnd?| (pf)
      (|pfAbSynOp?| pf '|And|))

———

defun Return the Left part of an And node

    — defun pfAndLeft 0 —

    (defun |pfAndLeft| (pf)
      (cadr pf))

———
defun Return the Right part of an And node

— defun pfAndRight 0 —

(defun |pfAndRight| (pf)
  (caddr pf))

__________

defun Flatten a list of lists

— defun pfAppend 0 —

(defun |pfAppend| (list)
  (apply #'append list))

__________

defun Is this an Application node?

[pfAbSynOp? p634]

— defun pfApplication? —

(defun |pfApplication?| (pf)
  (|pfAbSynOp?| pf ’|Application|))

__________

defun Create an Assign node

[pfTree p482]

— defun pfAssign —

(defun |pfAssign| (pflhsitems pfrhs)
  (|pfTree| ’|Assign| (list pflhsitems pfrhs)))

__________
defun Is this an Assign node?

[pfAbSynOp? p634]

— defun pfAssign? —

(defun |pfAssign?| (pf)
  (|pfAbSynOp?| pf '|Assign|))

defun Return the parts of an LhsItem of an Assign node

[pfParts p479]
[pfAssignLhsItems p486]

— defun pf0AssignLhsItems 0 —

(defun |pf0AssignLhsItems| 0 (pf)
  (|pfParts| (|pfAssignLhsItems| pf)))

defun Return the LhsItem of an Assign node

— defun pfAssignLhsItems 0 —

(defun |pfAssignLhsItems| 0 (pf)
  (cadr pf))

defun Return the RHS of an Assign node

— defun pfAssignRhs 0 —

(defun |pfAssignRhs| 0 (pf)
  (caddr pf))
defun Construct an application node for a brace

(defun pfBrace (a part)
  (list pfApplication (list pfIdPos '{} (list tokPosn a)) part))

---

defun Construct an Application node for brace-bars

(defun pfBraceBar (a part)
  (list pfApplication (list pfIdPos '{|{||} (list tokPosn a)) part))

---

defun Construct an Application node for a bracket

(defun pfBracket (a part)
  (list pfApplication (list pfIdPos '[] (list tokPosn a)) part))

---

defun Construct an Application node for bracket-bars


— defun pfBracketBar —

(defun pfBracketBar (a part)
  (pfApplication (pfIdPos '[] (tokPosn a)) part))

— defun Create a Break node —

(defun pfBreak (pffrom)
  (pfTree 'Break (list pffrom)))

— defun Is this a Break node? —

(defun pfBreak? (pf)
  (pfAbSynOp? pf 'Break)))

— defun Return the From part of a Break node —

(defun pfBreakFrom 0
  (cadr pf))
defun Construct a Coerceto node
[pfTree p482]

— defun pfCoerceto —

(defun |pfCoerceto| (pfexpr pftype)
  (|pfTree| |Coerceto| (list pfexpr pftype)))

---

defun Is this a CoerceTo node?
[pfAbSynOp? p634]

— defun pfCoerceto? —

(defun |pfCoerceto?| (pf)
  (|pfAbSynOp?| pf |Coerceto|))

---

defun Return the Expression part of a CoerceTo node

— defun pfCoercetoExpr 0 —

(defun |pfCoercetoExpr| (pf)
  (cadr pf))

---

defun Return the Type part of a CoerceTo node

— defun pfCoercetoType 0 —

(defun |pfCoercetoType| (pf)
  (caddr pf))

---
defun Return the Body of a Collect node

— defun pfCollectBody 0 —

(defun |pfCollectBody| (pf)
  (cadr pf))

---

defun Return the Iterators of a Collect node

— defun pfCollectIterators 0 —

(defun |pfCollectIterators| (pf)
  (caddr pf))

---

defun Create a Collect node
[pfTree p482]

— defun pfCollect —

(defun |pfCollect| (pfbody pfiterators)
  (|pfTree| ’|Collect| (list pfbody pfiterators)))

---

defun Is this a Collect node?
[pfAbSynOp? p634]

— defun pfCollect? —

(defun |pfCollect?| (pf)
  (|pfAbSynOp?| pf ’|Collect|))

---
14.5. TREES

defun pfDefinition
[pfTree p482]
— defun pfDefinition —
(defun |pfDefinition| (pfLhsItems pfrhs)
  (|pfTree| 'Definition| (list pfLhsItems pfrhs)))

defun Return the Lhs of a Definition node
— defun pfDefinitionLhsItems 0 —
(defun |pfDefinitionLhsItems| (pf)
  (cadr pf))

defun Return the Rhs of a Definition node
— defun pfDefinitionRhs 0 —
(defun |pfDefinitionRhs| (pf)
  (caddr pf))

defun Is this a Definition node?
[pfAbSynOp? p634]
— defun pfDefinition? —
(defun |pfDefinition?| (pf)
  (|pfAbSynOp?| pf 'Definition|)))
defun Return the parts of a Definition node

    defun pf0DefinitionLhsItems —
    (defun pf0DefinitionLhsItems (pf) (list pfParts pfDefinitionLhsItems pf))

defun Create a Do node

    defun pfDo —
    (defun pfDo (pfbody) (list pfTree 'Do (list pfbody)))

defun Is this a Do node?

    defun pfDo? —
    (defun pfDo? (pf) (list pfAbSynOp 'Do pf_tree))

defun Return the Body of a Do node

    defun pfDoBody 0 —
    (defun pfDoBody (pf) (cadr pf))
defun Construct a Sequence node

(pfTuple p522)
(pfListOf p475)
(pfSequence p517)

— defun pfEnSequence —

(defun |pfEnSequence| (a)
  (cond
    ((null a) (|pfTuple| (|pfListOf| a)))
    ((null (cdr a)) (car a))
    (t (|pfSequence| (|pfListOf| a))))

———

defun Construct an Exit node

(pfTree p482)

— defun pfExit —

(defun |pfExit| (pfcond pfexpr)
  (|pfTree| '|Exit| (list pfcond pfexpr)))

———

defun Is this an Exit node?

(pfAbSynOp? p634)

— defun pfExit? —

(defun |pfExit?| (pf)
  (|pfAbSynOp?| pf '|Exit|))

———

defun Return the Cond part of an Exit

— defun pfExitCond 0 —
(defun pfExitCond (pf)
  (cadr pf))

---

defun Return the Expression part of an Exit

    — defun pfExitExpr 0 —

(defun pfExitExpr (pf)
  (caddr pf))

---

defun Create a Export node

    — defun pfExport —

(defun pfExport (pfitems)
  (pfTree 'Export (list pfitems)))

---

defun Construct an Expression leaf node

    — defun pfExpression —

(defun pfExpression (expr &rest optpos)
  (pfLeaf 'expression expr (ifcar optpos)))

---

defun pfFirst

    — defun pfFirst 0 —
14.5. TREES

(defun |pfFirst| (form)
  (cadr form))

defun Create an Application Fix node
[pfApplication p483]
[pfId p476]

  — defun pfFix —

(defun |pfFix| (pf)
  (|pfApplication| (|pfId| 'Y) pf))

(defun Create a Free node
[pfTree p482]

  — defun pfFree —

(defun |pfFree| (pfitems)
  (|pfTree| '|Free| (list pfitems)))

(defun Is this a Free node?
[pfAbSynOp? p634]

  — defun pfFree? —

(defun |pfFree?| (pf)
  (|pfAbSynOp?| pf '|Free|))
defun Return the parts of the Items of a Free node

(defun pf0FreeItems |pfParts| (pf) (|pfParts| (|pfFreeItems| pf)))

defun Return the Items of a Free node

(defun pfFreeItems 0 (pf) (cadr pf))

defun Construct a ForIn node

(defun pfForin |pfTree| 'Forin (list pflhs pfwhole))

defun Is this a ForIn node?

(defun pfForin? (pf) (|pfAbSynOp?| pf 'Forin))
14.5. TREES

defun Return all the parts of the LHS of a ForIn node

(defun pf0ForinLhs (pf)
  (pfParts (pfForinLhs pf)))

— defun pf0ForInLhs —

defun Return the LHS part of a ForIn node

— defun pfForinLhs 0 —

(defun pfForinLhs (pf)
  (cadr pf))

— defun pfForinLhs —

defun Return the Whole part of a ForIn node

— defun pfForinWhole 0 —

(defun pfForinWhole (pf)
  (caddr pf))

— defun pfForinWhole —

defun pfFromDom

(defun pfFromDom
  (pfApplication? pfApplication pfApplicationOp pfApplicationArg pfFromDom pf))

— defun pfFromDom —
(defun |pfFromDom| (dom expr)
  (cond
    ((|pfApplication?| expr)
     (|pfApplication|
      (|pfFromdom| (|pfApplicationOp| expr) dom)
      (|pfApplicationArg| expr)))
    (t (|pfFromdom| expr dom))))

defun Construct a Fromdom node

[pfTree p482]

— defun pfFromdom —

(defun |pfFromdom| (pfwhat pfdomain)
  (|pfTree| '|Fromdom| (list pfwhat pfdomain)))

defun Is this a Fromdom mode?

[pfAbSynOp? p634]

— defun pfFromdom? —

(defun |pfFromdom?| (pf)
  (|pfAbSynOp?| pf '|Fromdom|))

defun Return the What part of a Fromdom node

— defun pfFromdomWhat 0 —

(defun |pfFromdomWhat| (pf)
  (cadr pf))
defun Return the Domain part of a Fromdom node

— defun pfFromdomDomain 0 —

(defun |pfFromdomDomain| (pf)
  (caddr pf))

defun Construct a Hide node

[pfTree p482]

— defun pfHide —

(defun |pfHide| (a part)
  (declare (ignore a))
  ([pfTree| ']Hide| (list part)))

defun pfIf

[pfTree p482]

— defun pfIf —

(defun |pfIf| (pfcond pfthen pfelse)
  ([pfTree| ']If| (list pfcond pfthen pfelse)))

defun Is this an If node?

[pfAbSynOp? p634]

— defun pfIf? —

(defun |pfIf?| (pf)
  ([pfAbSynOp?| pf |'If|]))
defun Return the Cond part of an If

— defun pfIfCond 0 —

(defun |pfIfCond| (pf)
  (cadr pf))

---------

defun Return the Then part of an If

— defun pfIfThen 0 —

(defun |pfIfThen| (pf)
  (caddr pf))

---------

defun pfIfThenOnly

[pfIf p499]
[pfNothing p475]

— defun pfIfThenOnly —

(defun |pfIfThenOnly| (pred cararg)
  (|pfIf| pred cararg (|pfNothing|)))

---------

defun Return the Else part of an If

— defun pfIfElse 0 —

(defun |pfIfElse| (pf)
  (cadddr pf))

---------
14.5. TREES

defun Construct an Import node

(defun |pfImport| (pfitems)
  (|pfTree| 'Import (list pfitems)))


defun Construct an Iterate node

(defun |pfIterate| (pffrom)
  (|pfTree| 'Iterate (list pffrom)))


defun Is this an Iterate node?

(defun |pfIterate?| (pf)
  (|pfAbSynOp?| pf 'Iterate))


defun Handle an infix application

(defun |pfInfApplication| — defun pfInfApplication —

(defun |pfListOf| p475)
  (|pfIdSymbol| p477)
  (|pfAnd| p483)
  (|pfOr| p510)
  (|pfApplication| p483)
  (|pfTuple| p522)

        — defun pfInfApplication —
(defun pfInfApplication (op left right)
  (cond
    ((eq (pfIdSymbol op) '|and|) (pfAnd left right))
    ((eq (pfIdSymbol op) '|or|) (pfOr left right))
    (t (pfApplication op (pfTuple (pfListOf (list left right)))))))

defun Create an Inline node

(defun pfInline (pfitems)
  (pfTree '|Inline| (list pfitems)))

defun pfLam

(defun pfLam (variable body)
  (let (bdy rets)
    (if (pfAbSynOp? body '|returntyped|)
      (setq rets (pfFirst body))
      (setq rets (pfNothing)))
    (if (pfAbSynOp? body '|returntyped|)
      (setq bdy (pfSecond body))
      (setq bdy body))
    (pfLambda variable rets bdy)))
14.5. TREES

defun pfLambda

[pfTree p482]

--- defun pfLambda ---

(defun |pfLambda| (pfargs pfrets pfbody)
  (|pfTree| '|Lambda| (list pfargs pfrets pfbody)))

-----

defun Return the Body part of a Lambda node

--- defun pfLambdaBody 0 ---

(defun |pfLambdaBody| (pf)
  (cadddr pf))

-----

defun Return the Rets part of a Lambda node

--- defun pfLambdaRets 0 ---

(defun |pfLambdaRets| (pf)
  (caddr pf))

-----

defun Is this a Lambda node?

[pfAbSynOp? p634]

--- defun pfLambda? ---

(defun |pfLambda?| (pf)
  (|pfAbSynOp?| pf '|Lambda|))

-----
defun Return the Args part of a Lambda node

--- defun pfLambdaArgs 0 ---

(defun |pfLambdaArgs| (pf)
  (cadr pf))

---

defun Return the Args of a Lambda Node

[|pfParts| p479]
[|pfLambdaArgs| p504]

--- defun pf0LambdaArgs ---

(defun |pf0LambdaArgs| (pf)
  (|pfParts| (|pfLambdaArgs| pf)))

---

defun Construct a Local node

[|pfTree| p482]

--- defun pfLocal ---

(defun |pfLocal| (pfitems)
  (|pfTree| 'Local (list pfitems)))

---

defun Is this a Local node?

[|pfAbSynOp?| p634]

--- defun pfLocal? ---

(defun |pfLocal?| (pf)
  (|pfAbSynOp?| pf 'Local)))

---
defun Return the parts of Items of a Local node

(defun pfParts (pf)
  (pfLocalItems pf))

defun Return the Items of a Local node

(defun pfLocalItems (pf)
  (cadr pf))

defun Construct a Loop node

(defun pfLoop (pfiterators)
  (pfTree 'Loop (list pfiterators)))

defun pfLoop1

(defun pfLoop1 (body)
  (pfLoop (pfListOf (list (pfDo body)))))
defun Is this a Loop node?

[pfAbSynOp? p634]

— defun pfLoop? —

(defun pfLoop? (pf)
  (pfAbSynOp? pf 'Loop))

—

defun Return the Iterators of a Loop node

— defun pfLoopIterators 0 —

(defun pfLoopIterators (pf)
  (cadr pf))

—

defun pf0LoopIterators

[pfParts p479]
[pf0LoopIterators p506]

— defun pf0LoopIterators —

(defun pf0LoopIterators (pf)
  (pfParts (pfLoopIterators pf)))

—

defun pfLp

[pfLoop p505]
[pfListOf p475]
[pfDo p492]

— defun pfLp —
(defun pfLp (iterators body)
  (pfLoop (pfListOf (append iterators (list (pfDo body))))))

defun Create a Macro node

[pfTree p482]

— defun pfMacro —

(defun pfMacro (pflhs pfrhs)
  (pfTree 'Macro (list pflhs pfrhs)))

— defun pfMacro? —

(defun pfMacro? (pf)
  (pfAbSynOp? pf 'Macro))

— defun pfMacroLhs 0 —

(defun pfMacroLhs (pf)
  (cadr pf))

— defun pfMacroRhs 0 —
(defun |pfMacroRhs| (pf) (caddr pf))

---

defun Construct an MLambda node

[pfTree p482]

---

---

---

defun Return the Args of an MLambda

---

---

---

defun Return the parts of an MLambda argument

[pfParts p479]
14.5. TREES

(defun pf0MLambdaArgs (pf)
  (|pfParts| (|pfMLambdaArgs| pf)))

defun pfMLambdaBody

  — defun pfMLambdaBody 0 —

(defun |pfMLambdaBody| (pf)
  (caddr pf))

defun Is this a Not node?

[|pfAbSynOp? p634|

  — defun pfNot? —

(defun |pfNot?| (pf)
  (|pfAbSynOp?| pf '|Not|)))

—

defun Return the Arg part of a Not node

  — defun pfNotArg 0 —

(defun |pfNotArg| (pf)
  (cadr pf))

—

defun Construct a NoValue node

[|pfTree p482|

  — defun pfNovalue —
(defun pfNovalue (pfexpr)
  (pfTree 'Novalue (list pfexpr)))

---

defun Is this a Novalue node?
[pfAbSynOp? p634]

  — defun pfNovalue? —

(defun pfNovalue? (pf)
  (pfAbSynOp? pf 'Novalue))

---

defun Return the Expr part of a Novalue node

  — defun pfNovalueExpr 0 —

(defun pfNovalueExpr 0 (pf)
  (cadr pf))

---

defun Construct an Or node
[pfTree p482]

  — defun pfOr —

(defun pfOr (pfleft pfright)
  (pfTree 'Or (list pfleft pfright)))

---

defun Is this an Or node?
[pfAbSynOp? p634]

  — defun pfOr? —
(defun |pfOr?| (pf)
  (|pfAbSynOp?| pf '|Or|))

---

defun Return the Left part of an Or node

   — defun pfOrLeft 0 —

(defun |pfOrLeft| (pf)
  (cadr pf))

---

defun Return the Right part of an Or node

   — defun pfOrRight 0 —

(defun |pfOrRight| (pf)
  (caddr pf))

---

defun Return the part of a parenthesised expression

   — defun pfParen —

(defun |pfParen| (a part)
  (declare (ignore a))
  part)

---

defun pfPretend

  [pfTree p482]

   — defun pfPretend —
(defun pfPretend (pfexpr pftype)
  (pftree 'Pretend (list pfexpr pftype)))

---

(defun Is this a Pretend node?

[pfAbSynOp? p634]

---

(defun pfPretend?
  (pfAbSynOp? pf 'Pretend))

---

(defun Return the Expression part of a Pretend node

---

(defun pfPretendExpr
  (cadr pf))

---

(defun Return the Type part of a Pretend node

---

(defun pfPretendType
  (caddr pf))

---

(defun Construct a QualType node

[pfTree p482]

---

(defun pfQualType ---
(defun |pfQualType| (pftype pfqual)
  (|pfTree| 'Qua|Type| (list pftype pfqual)))

---

defun Construct a Restrict node

[pfTree p482]

    — defun pfRestrict —

(defun |pfRestrict| (pfexpr pftype)
  (|pfTree| 'Restrict| (list pfexpr pftype)))

---

defun Is this a Restrict node?

[pfAbSynOp? p634]

    — defun pfRestrict? —

(defun |pfRestrict?| (pf)
  (|pfAbSynOp?| pf 'Restrict)))

---

defun Return the Expr part of a Restrict node

    — defun pfRestrictExpr 0 —

(defun |pfRestrictExpr| (pf)
  (cadr pf))

---

defun Return the Type part of a Restrict node

    — defun pfRestrictType 0 —
(defun pfRestrictType (pf)
  (caddr pf))

---

defun Construct a RetractTo node

[pfTree p482]

---

defun pfRetractTo

(defun pfRetractTo (pfexpr pftype)
  (|pfTree| '|RetractTo| (list pfexpr pftype)))

---

defun Construct a Return node

[pfTree p482]

---

defun pfReturn

(defun pfReturn (pfexpr pffrom)
  (|pfTree| '|Return| (list pfexpr pffrom)))

---

defun Is this a Return node?

[pfAbSynOp? p634]

---

defun pfReturn?

(defun pfReturn? (pf)
  (|pfAbSynOp?| pf '|Return|))

---

defun Return the Expr part of a Return node

---

defun pfReturnExpr
(defun pfReturnExpr (pf)
  (cadr pf))

---

defun pfReturnNoName

[pfReturn p514]
[pfNothing p475]

---

defun pfReturnNoName

(defun pfReturnNoName (value)
  (pfReturn value (pfNothing)))

---

defun Construct a ReturnTyped node

[pfTree p482]

---

defun pfReturnTyped

(defun pfReturnTyped (type body)
  (pfTree 'returntyped (list type body)))

---

defun Construct a Rule node

[pfTree p482]

---

defun pfRule

(defun pfRule (pflhsitems pfrhs)
  (pfTree 'Rule (list pflhsitems pfrhs)))

---
defun Return the Lhs of a Rule node

— defun pfRuleLhsItems 0 —

(defun |pfRuleLhsItems| (pf)
  (cadr pf))

defun Return the Rhs of a Rule node

— defun pfRuleRhs 0 —

(defun |pfRuleRhs| (pf)
  (caddr pf))

defun Is this a Rule node?

[pfAbSynOp? p634]

— defun pfRule? —

(defun |pfRule?| (pf)
  (|pfAbSynOp?| pf '|Rule|))

defun pfSecond

— defun pfSecond 0 —

(defun |pfSecond| (form)
  (caddr form))
defun Construct a Sequence node
[pfTree p482]

— defun pfSequence —

(defun |pfSequence| (pfargs)
  (|pfTree| |Sequence| (list pfargs)))

—

defun Return the Args of a Sequence node

— defun pfSequenceArgs 0 —

(defun |pfSequenceArgs| (pf)
  (cadr pf))

—

defun Is this a Sequence node?

[ pfAbSynOp? p634 ]

— defun pfSequence? —

(defun |pfSequence?| (pf)
  (|pfAbSynOp?| pf |Sequence|))

—

defun Return the parts of the Args of a Sequence node

[ pfParts p479]
[ pfSequenceArgs p517]

— defun pf0SequenceArgs —

(defun |pf0SequenceArgs| (pf)
  (|pfParts| (|pfSequenceArgs| pf)))

—
defun Create a Suchthat node

(defun pfSuchthat (pfcond)
  (list pfTree 'Suchthat (list pfcond)))

defun Is this a SuchThat node?

(defun pfSuchthat? (pf)
  (pfAbSynOp? pf 'Suchthat))

defun Return the Cond part of a SuchThat node

(defun pfSuchthatCond (pf)
  (cadr pf))

defun Create a Tagged node

(defun pfTagged (pftag pfexpr)
  (list pfTree 'Tagged (list pftag pfexpr)))
defun Is this a Tagged node?
[pfAbSynOp? p634]
— defun pfTagged? —
(defun |pfTagged?| (pf)
  (|pfAbSynOp?| pf 'Tagged)))

— defun Return the Expression portion of a Tagged node —
(defun |pfTaggedExpr| (pf)
  (caddr pf))

— defun Return the Tag of a Tagged node —
(defun |pfTaggedTag| (pf)
  (cadr pf))

— defun pfTaggedToTyped —
[pfTagged? p519]
[pfTaggedExpr p519]
[pfNothing p475]
[pfTaggedTag p519]
[pfId? p476]
[pfId p476]
[pfTyped p520]
[pfSuch p474]
[pfInfApplication p501]
--- defun pfTaggedToTyped ---

(defun pfTaggedToTyped (arg)
  (let ((a form rt))
    (if (pfTagged? arg)
        (setq rt (pfTaggedExpr arg))
        (setq rt (pfNothing)))
    (if (pfTagged? arg)
        (setq form (pfTaggedTag arg))
        (setq form arg))
    (cond
      ((null (pfId? form))
        (setq a (pfId (gensym)))
        (pfTyped (pfSuch a (pfInfApplication (pfId '=') a form)) rt))
      (t (pfTyped form rt))))

---

defun pfTweakIf

[pfIfElse p500]
[pfNothing? p475]
[pfListOf p475]
[pfTree p482]
[pfIfCond p500]
[pfIfThen p500]

--- defun pfTweakIf ---

(defun pfTweakIf (form)
  (let (b a)
    (setq a (pfIfElse form))
    (setq b (if (pfNothing? a) (pfListOf NIL) a))
    (pfTree '|WIf| (list (pfIfCond form) (pfIfThen form) b))))

---

defun Construct a Typed node

[pfTree p482]

--- defun pfTyped ---

(defun pfTyped (pfid pftype)
14.5. TREES

(defun Is this a Typed node?

[pfAbSynOp? p634]

— defun pfTyped? —

(defun |pfTyped?| (pf)
  (|pfAbSynOp?| pf '|Typed|))

———

defun Return the Type of a Typed node

— defun pfTypedType 0 —

(defun |pfTypedType| (pf)
  (caddr pf))

———

defun Return the Id of a Typed node

— defun pfTypedId 0 —

(defun |pfTypedId| (pf)
  (cadr pf))

———

defun Construct a Typing node

[pfTree p482]

— defun pfTyping —
(defun |pfTyping| (pfitems)
  (|pfTree| '|Typing| (list pfitems)))

---

defun Return a Tuple node

[|pfTree| p482]

— defun pfTuple —

(defun |pfTuple| (pfparts)
  (|pfTree| '|Tuple| (list pfparts)))

---

defun Return a Tuple from a List

[|pfTuple| p522]
[|pfListOf| p475]

— defun pfTupleListOf —

(defun |pfTupleListOf| (pfparts)
  (|pfTuple| (|pfListOf| pfparts)))

---

defun Is this a Tuple node?

[|pfAbSynOp?| p634]

— defun pfTuple? —

(defun |pfTuple?| (pf)
  (|pfAbSynOp?| pf '|Tuple|))

---
defun Return the Parts of a Tuple node

defun pfTupleParts 0 —

(defun |pfTupleParts| (pf)
  (cadr pf))

—

defun Return the parts of a Tuple

[pfParts p479]
[pfTupleParts p523]

— defun pf0TupleParts —

(defun |pf0TupleParts| (pf)
  (|pfParts| (|pfTupleParts| pf)))

—

defun Return a list from a Sequence node

[pfSequence? p517]
[pfAppend p485]
[pf0SequenceArgs p517]
[pfListOf p475]

— defun pfUnSequence —

(defun |pfUnSequence| (x)
  (if (|pfSequence?| x)
    (|pfList0f| (|pfAppend| (|pf0SequenceArgs| x)))
    (|pfList0f| x)))

—

defun The comment is attached to all signatutres

[pfWDeclare p524]
[pfParts p479]
--- defun pfWDec ---
(defun |pfWDec| (doc name)
  (mapcar #'(lambda (i) (|pfWDeclare| i doc)) (|pfParts| name)))

------

defun Construct a WDeclare node

[pfTree p482]

--- defun pfWDeclare ---
(defun |pfWDeclare| (pfsignature pfdoc)
  (|pfTree| '|WDeclare| (list pfsignature pfdoc)))

------

defun Construct a Where node

[pfTree p482]

--- defun pfWhere ---
(defun |pfWhere| (pfcontext pfexpr)
  (|pfTree| '|Where| (list pfcontext pfexpr)))

------

defun Is this a Where node?

[pfAbSynOp? p634]

--- defun pfWhere? ---
(defun |pfWhere?| (pf)
  (|pfAbSynOp?| pf '|Where|))

------
14.5. TREES

defun Return the parts of the Context of a Where node

[pfParts p179]
[pfWhereContext p525]

— defun pf0WhereContext —

(defun |pf0WhereContext| (pf)
  (|pfParts| (|pfWhereContext| pf)))

———

defun Return the Context of a Where node

— defun pfWhereContext 0 —

(defun |pfWhereContext| (pf)
  (cadr pf))

———

defun Return the Expr part of a Where node

— defun pfWhereExpr 0 —

(defun |pfWhereExpr| (pf)
  (caddr pf))

———

defun Construct a While node

[pfTree p482]

— defun pfWhile —

(defun |pfWhile| (pfcond)
  (|pfTree| '|While| (list pfcond)))

———
defun Is this a While node?
[pfAbSynOp p634]

<table>
<thead>
<tr>
<th>defun pfWhile?</th>
</tr>
</thead>
<tbody>
<tr>
<td>(defun</td>
</tr>
<tr>
<td>(pfAbSynOp</td>
</tr>
</tbody>
</table>

defun Return the Cond part of a While node

<table>
<thead>
<tr>
<th>defun pfWhileCond 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(defun</td>
</tr>
<tr>
<td>(cadr pf))</td>
</tr>
</tbody>
</table>

defun Construct a With node
[pfTree p482]

<table>
<thead>
<tr>
<th>defun pfWith</th>
</tr>
</thead>
<tbody>
<tr>
<td>(defun</td>
</tr>
<tr>
<td>(pfTree</td>
</tr>
</tbody>
</table>

defun Create a Wrong node
[pfTree p482]

<table>
<thead>
<tr>
<th>defun pfWrong</th>
</tr>
</thead>
<tbody>
<tr>
<td>(defun</td>
</tr>
<tr>
<td>(pfTree</td>
</tr>
</tbody>
</table>


14.5. TREES

defun Is this a Wrong node?
[pfAbSynOp? p634]

--- defun pfWrong? ---

(defun |pfWrong?| (pf)
  (|pfAbSynOp?| pf 'Wrong)))
Chapter 15

Pftree to s-expression translation

Pftree to s-expression translation. Used to interface the new parser technology to the interpreter. The input is a parseTree and the output is an old-parser-style s-expression.

defun Pftree to s-expression translation

[pf2Sex1 p530]
[$insideSEQ p??]
[$insideApplication p??]
[$insideRule p??]
[$QuietCommand p271]

— defun pf2Sex —

(defun pf2Sex (pf)
(let ([insideSEQ] [insideApplication] [insideRule])
(declare (special insideSEQ insideApplication insideRule)
[QuietCommand]))
(setq [QuietCommand] nil)
(setq [insideRule] nil)
(setq [insideApplication] nil)
(setq [insideSEQ] nil)
([pf2Sex1 pf]))

———
defun Pftree to s-expression translation inner function

[pfNothing? p475]
[pfSymbol? p482]
[pfSymbolSymbol p482]
[plLiteral? p478]
[plLiteral2Sex p534]
[pfIdSymbol p477]
[pfApplication? p485]
[pfApplication2Sex p535]
[pfTuple? p522]
[pf2Sex1 p530]
[pf0TupleParts p523]
[pff? p499]
[pffCond p500]
[pffThen p500]
[pffElse p500]
[pfTagged? p519]
[pfTaggedTag p519]
[pfTaggedExpr p519]
[pfCoerceto? p489]
[pfCoercetoExpr p489]
[pfCoercetoType p489]
[pfPretend? p512]
[pfPretendExpr p512]
[pfPretendType p512]
[pfFromdom? p498]
[opTran p553]
[pfFromdomWhat p498]
[pfFromdomDomain p499]
[pfSequence? p517]
[pfSequence2Sex p540]
[pfExit? p493]
[pfExitCond p493]
[pfExitExpr p494]
[pfLoop? p506]
[loopIters2Sex p541]
[pf0LoopIterators p506]
[pfCollect? p490]
[pfCollect2Sex p544]
[pfForin? p496]
[pf0ForinLhs p497]
[pfForinWhole p497]
[pfWhile? p526]
[pfWhileCond p526]
[pfSuchthat? p518]
[keyedSystemError p??]
[pfSuchthatCond p518]
[pfDo? p492]
[pfDoBody p492]
[pfTyped? p521]
[pfTypedType p521]
[pfTypedId p521]
[pfAssign? p486]
[pfAssignLhsItems p486]
[pfAssignRhs p486]
[pfDefinition? p491]
[pfDefinition2Sex p545]
[pfLambda? p503]
[pfLambda2Sex p548]
[pfMLambda? p508]
[pfRestrict? p513]
[pfRestrictExpr p513]
[pfRestrictType p513]
[pfFree? p495]
[pfFreeItems p496]
[pfLocal? p504]
[pfLocalItems p505]
[pfWrong? p527]
[spadThrow p??]
[pfAnd? p484]
[pfAndLeft p484]
[pfAndRight p485]
[pfOr? p510]
[pfOrLeft p511]
[pfOrRight p511]
[pfNot? p509]
[pfNotArg p509]
[pfNovalue? p510]
[pfNovalueExpr p510]
[pfRule? p516]
[pfRule2Sex p548]
[pfBreak? p488]
[pfBreakFrom p488]
[pfMacro? p507]
[pfReturn? p514]
[pfReturnExpr p514]
[pfIterate? p501]
[pfWhere? p524]
[pfWhereContext p525]
[pfWhereExpr p525]
[pfAbSynOp p634]
(defun pf2Sex1 (pf)
  (let (args idList type op tagPart tag s)
    (declare (special $insideSEQ $insideRule $QuietCommand))
    (cond
      ((pfNothing? pf) '|noBranch|)
      ((pfSymbol? pf)
        (if (eq $insideRule '|left|)
            (progn
              (setq s (pfSymbolSymbol pf))
              (list '|constant (list 'quote s)))
            (list 'quote (pfSymbolSymbol pf))))
      ((pfLiteral? pf) (pfLiteral2Sex pf))
      ((pfId? pf)
        (if $insideRule
            (progn
              (setq s (pfIdSymbol pf))
              (if (member s '(|%pi| |%e| |%i|))
                  s
                  (list 'quote s)))
            (pfIdSymbol pf)))
      ((pfApplication? pf) (pfApplication2Sex pf))
      ((pfTuple? pf) (cons '|Tuple (mapcar #'pf2Sex1 (pf0TupleParts pf))))
      ((pfIf? pf)
        (list 'if (pf2Sex1 (pfIfCond pf))
                 (pf2Sex1 (pfIfThen pf))
                 (pf2Sex1 (pfIfElse pf))))
      ((pfTagged? pf)
        (setq tag (pfTaggedTag pf))
        (setq tagPart
          (if (pfTuple? tag)
              (cons '|Tuple (mapcar '#\'pf2Sex1 (pf0TupleParts tag)))
              (pf2Sex1 tag))
          (list '|: tagPart (pf2Sex1 (pfTaggedExpr pf))))
      ((pfCoerceTo? pf)
        (list '|:: (pf2Sex1 (pfCoerceToExpr pf))
              (pf2Sex1 (pfCoerceToType pf))))
      ((pfPretend? pf)
        (list '|pretend (pf2Sex1 (pfPretendExpr pf))
              (pf2Sex1 (pfPretendType pf))))
      ((pfFromDom? pf)
        (setq op (opTran (pf2Sex1 (pfFromDomWhat pf))))
        (when (eq op '|braceFromCurly|) (setq op '|seq|)))
    )))
(list '|$elt| (|pf2Sex1| (|pfFromdomDomain| pf)) op))
((|pfSequence?| pf) (|pfSequence2Sex| pf))
((|pfExit?| pf)
 (if |$insideSEQ|
 (list '|exit| (|pf2Sex1| (|pfExitCond| pf)))
 (|pf2Sex1| (|pfExitExpr| pf))))
((|pfLoop?| pf) (cons 'repeat (|loopIters2Sex| (|pf0LoopIterators| pf))))
((|pfCollect?| pf) (|pfCollect2Sex| pf))
((|pfForin?| pf)
 (cons 'in
 (append (mapcar #'|pf2Sex1| (|pf0ForinLhs| pf))
 (list (|pf2Sex1| (|pfForinWhole| pf)))))
((|pfWhile?| pf) (list 'while (|pf2Sex1| (|pfWhileCond| pf)))))
((|pfSuchthat?| pf)
 (if (eq |$insideRule| '|left|)
 (keyedSystemError| "Unexpected error in call to system function %1"
 (list "pf2Sex1: pfSuchThat"))
 (list '|\| (|pf2Sex1| (|pfSuchthatCond| pf)))))
((|pfDo?| pf) (|pf2Sex1| (|pfDoBody| pf)))
((|pfTyped?| pf)
 (setq type (|pfTypedType| pf))
 (if (|pfNothing?| type)
 (|pf2Sex1| (|pfTypedId| pf))
 (list '|' (|pf2Sex1| (|pfTypedId| pf)) (|pf2Sex1| (|pfTypedType| pf)))))
((|pfAssign?| pf)
 (setq idList (mapcar #'|pf2Sex1| (|pf0AssignLhsItems| pf)))))
((|pfDefinition?| pf)
 (if (not (eql (length idList) 1))
 (setq idList (cons '|Tuple| idList))
 (setq idList (car idList))
 (list 'let idList (|pfAssignRhs| pf))))
((|pfDefinition2Sex| pf)
 (if (|pfNothing?| (|pfDefinition2Sex| pf)) (|pfAssign2Sex| pf)))
((|pfLambda?| pf) (|pfLambda2Sex| pf))
((|pfMLambda?| pf) '|/throwAway|)
((|pfRestrict?| pf)
 (list '0 (|pfRestrictExpr| pf))
 (|pfRestrictType| pf)))
((|pfFree?| pf) (cons '|free| (mapcar #'|pf2Sex1| (|pf0FreeItems| pf)))))
((|pfLocal?| pf) (cons '|local| (mapcar #'|pf2Sex1| (|pf0LocalItems| pf)))))
((|pfWrong?| pf) (|spadThrow|))
((|pfAnd?| pf)
 (list 'and (|pf2Sex1| (|pfAndLeft| pf))
 (|pf2Sex1| (|pfAndRight| pf)))))
((|pfOr?| pf)
 (list 'or (|pf2Sex1| (|pfOrLeft| pf))
 (|pf2Sex1| (|pfOrRight| pf)))))
((|pfNot?| pf) (list 'not (|pf2Sex1| (|pfNotArg| pf)))))
((|pfNovalue?| pf)
 (setq |$QuietCommand| t))
defun Convert a Literal to an S-expression

(defun pfLiteral2Sex (pf)
  (let (s type)
    (declare (special $insideRule))
    (setq type (pfLiteralClass pf))
    (cond
      ((eq type '|integer|) (read-from-string (pfLiteralString pf)))
      ((or (eq type '|string|) (eq type '|char|))
       (pfLiteralString pf))
      ((eq type '|float|) (float2Sex (pfLiteralString pf)))
      ((eq type '|symbol|)
       (if $insideRule
         (progn
           (setq s (pfSymbolSymbol pf))
           (list 'quote s))
         (pfSymbolSymbol pf)))
      ))
(eq type 'expression) (list 'quote (pfLeafToken pf)))
(t
 (keyedSystemError "Unexpected error in call to system function %1"
 (list "pfLiteral2Sex: unexpected form"))))

---

defun Convert a float to an S-expression

[$useBFasDefault p??]

--- defun float2Sex ---

(defun float2Sex (num)
  (let (exp frac bfForm fracPartString intPart dotIndex expPart mantPart eIndex)
    (declare (special $useBFasDefault))
    (setq eIndex (search "e" num))
    (if eIndex
      (setq mantPart (subseq num 0 eIndex))
      (setq mantPart num))
    (if eIndex
      (setq expPart (read-from-string (subseq num (+ eIndex 1))))
      (setq expPart 0))
    (setq dotIndex (search "." mantPart))
    (if dotIndex
      (setq intPart (read-from-string (subseq mantPart 0 dotIndex)))
      (setq intPart (read-from-string mantPart)))
    (if dotIndex
      (setq fracPartString (subseq mantPart (+ dotIndex 1)))
      (setq fracPartString 0))
    (setq bfForm
      (make-float intPart (read-from-string fracPartString)
                   (length fracPartString) expPart))
    (if $useBFasDefault
      (progn
        (setq frac (cadr bfForm))
        (setq exp (cddr bfForm))
        (list (list '|elt| (list '|Float|) '|float|) frac exp 10))
      bfForm)))

---

defun Change an Application node to an S-expression

[pfOp2Sex p538]
[pfApplicationOp p484]
— defun pfApplication2Sex —

(defun pfApplication2Sex (pf)
  (let ((insideApplication x) val realOp tmp1 qt argSex typeList args op)
    (declare (special insideApplication insideRule))
    (setq insideApplication t)
    (setq op (pfOp2Sex (pfApplicationOp pf)))
    (setq op (opTran op))
    (cond
      ((eq op '->)
       (setq args (pf0TupleParts (pfApplicationArg pf)))
       (if (pfTuple? (car args))
         (setq typeList (mapcar 'pf2Sex1 (pf0TupleParts (car args))))
         (setq typeList (list (pf2Sex1 (car args))))
       (setq args (cons (pf2Sex1 (cadr args)) typeList))
       (cons 'Mapping args))
      ((and (eq op ':) (eq insideRule 'left))
       (list 'multiple (pf2Sex (pfApplicationArg pf))))
      ((and (eq op '?) (eq insideRule 'left))
       (list 'optional (pf2Sex (pfApplicationArg pf))))
      (t
       (setq args (pfApplicationArg pf))
       (cond
         ((pfTuple? args)
          (if (and (eq op '||) (eq $insideRule 'left))
            (pfSuchThat2Sex args)
            (progn
              (setq argSex (cdr (pf2Sex1 args)))
              (cond
                ((eq op '>) (list '< (cadr argSex) (car argSex)))
                ((eq op '>=) (list 'not (list '< (car argSex) (cadr argSex))))
                ((eq op '<=) (list 'not (list '< (cadr argSex) (car argSex))))
                ((eq op 'and) (list 'and (cadr (car argSex)) (cadr argSex)))
                ((eq op 'or) (list 'or (cadr (car argSex)) (cadr argSex)))
                ((eq op '\Iterate) (list 'iterate))
                ((eq op 'by) (cons 'by argSex))
                (eq op '\braceFromCurly)
                (if (and (consp argSex) (eq (car argSex) 'seq))
                  ...)
argSex
(cons 'seq argSex))
((and (consp op)
 (progn
  (setq qt (car op))
  (setq tmp1 (cdr op))
  (and (consp tmp1)
    (eq (cdr tmp1) nil)
    (progn
      (setq realOp (car tmp1))
      t)))))
  (eq qt 'quote))
(cons '|applyQuote| (cons op argSex)))
((setq val (|hasOptArgs?| argSex)) (cons op val))
(t (cons op argSex)))))))
((and (consp op)
   (progn
    (setq qt (car op))
    (setq tmp1 (cdr op))
    (and (consp tmp1)
     (eq (cdr tmp1) NIL)
    (progn
      (setq realOp (car tmp1))
      t)))
  (eq qt 'quote))
(list '|applyQuote| op (|pf2Sex1| args)))
((eq op '|braceFromCurly|)
 (setq x (|pf2Sex1| args))
 (if (and (consp x) (eq (car x) 'seq))
  x
  (list 'seq x)))
((eq op '|by|) (list 'by (|pf2Sex1| args)))
(t (list op (|pf2Sex1| args)))))

defun Convert a SuchThat node to an S-expression

defun pfSuchThat2Sex | @defun pfSuchThat2Sex |
defun pfSuchThat2Sex (args)
(let (rhsSex lhsSex argList name)
(declare (special |$predicateList|))
(setq name (gentemp))
(setq argList (|pf0TupleParts| args))
(setq lhsSex (|pf2Sex1| (car argList)))
(setq rhsSex (|pf2Sex| (cadr argList)))
(setq |$predicateList|
  (cons (cons name (cons lhsSex rhsSex)) |$predicateList|))

(defun pfOp2Sex
  [pf2Sex1 p530]
  [pmDontQuote? p539]
  [pfSymbol? p482]
  [quotedOpList p??]
  [insideRule p??]
  — defun pfOp2Sex —

(defun |pfOp2Sex| (pf)
  (let (realOp tmp1 op alreadyQuoted)
    (declare (special |$quotedOpList| |$insideRule|))
    (setq alreadyQuoted (|pfSymbol?| pf))
    (setq op (|pf2Sex1| pf))
    (cond
      ((and (consp op)
        (eq (car op) 'quote)
        (progn
          (setq tmp1 (cdr op))
          (and (consp tmp1)
            (eq (cdr tmp1) nil)
            (progn
              (setq realOp (car tmp1)) t))))
      (cond
        ((eq |$insideRule| '|left|) realOp)
        ((eq |$insideRule| '|right|)
          (cond
            (((pmDontQuote?| realOp) realOp)
             t
             (setq |quotedOpList| (cons op |quotedOpList|))
             op))
            ((eq realOp '\|\|) realOp)
            ((eq realOp '\|:|) realOp)
            ((eq realOp '?) realOp)
            (t op))))
defun pmDontQuote?

— defun pmDontQuote? 0 —

(defun pmDontQuote? (sy)
  (member sy '(+ - * ** ^ / |log| |exp| |pi| |sqrt| |ei| |li| |erf| |ci|
    |si| |dilog| |sin| |cos| |tan| |cot| |sec| |csc| |asin|
    |acos| |atan| |acot| |asec| |acsc| |sinh| |cosh| |tanh|
    |coth| |sech| |csch| |asinh| |acosh| |atanh| |acoth|
    |asech| |acsc))))

defun hasOptArgs?

— defun hasOptArgs? 0 —

(defun hasOptArgs? (argSex)
  (let (rhs lhs opt nonOpt tmp1 tmp2)
    (dolist (arg argSex)
      (cond
        ((and (consp arg)
           (eq (car arg) 'optarg)
           (progn
             (setq tmp1 (cdr arg))
             (and (consp tmp1)
               (progn
                 (setq lhs (car tmp1))
                 (setq tmp2 (cdr tmp1))
                 (and (consp tmp2)
                   (eq (cdr tmp2) nil)
                   (progn
                     (setq rhs (car tmp2))
                     t)))))
        (setq opt (cons (list lhs rhs) opt))))
    (t (setq nonOpt (cons arg nonOpt)))))
  (when opt
    (nconc (nreverse nonOpt) (list (cons '|construct| (nreverse opt)))))))
defun Convert a Sequence node to an S-expression

[pf2Sex1 p530]
[pf0SequenceArgs p517]
[$insideSEQ p??]

— defun pfSequence2Sex —

(defun |pfSequence2Sex| (pf)
  (let ((|$insideSEQ| tmp1 ruleList seq)
        (declare (special |$insideSEQ|)))
    (setq |$insideSEQ| t)
    (setq seq (|pfSequence2Sex0| (mapcar #'|pf2Sex1| (|pf0SequenceArgs| pf))))
    (cond
      ((and (consp seq)
            (eq (car seq) 'seq)
            (progn (setq ruleList (cdr seq)) 't)
            (consp ruleList)
            (progn
              (setq tmp1 (car ruleList))
              (and (consp tmp1) (eq (car tmp1) '|rule|)))
           (list '|ruleset| (cons '|construct| ruleList))))
    (t seq)))

defun pfSequence2Sex0

TPDHERE: rewrite this using (dolist (item seqList)...)

;pfSequence2Sex0 seqList ==
; null seqList => "noBranch"
; seqTranList := []
; while seqList != nil repeat
;   item := first seqList
;   item is ["exit", cond, value] =>
;   item := ["IF", cond, value, pfSequence2Sex0 rest seqList]
;   seqTranList := [item, :seqTranList]
;   seqList := rest seqList
; #seqTranList = 1 => first seqTranList
; ["SEQ", :nreverse seqTranList]
--- defun pfSequence2Sex0 ---

(defun |pfSequence2Sex0| (seqList)
  (let (value tmp2 cond tmp1 item seqTranList)
    (if (null seqList)
      '|noBranch|
      (progn
        ((lambda ()
          (loop
            (if (not seqList)
                (return nil)
                (progn
                  (setq item (car seqList))
                  (cond
                    ((and (consp item)
                        (eq (car item) '|exit|)
                        (progn
                          (setq tmp1 (cdr item))
                          (and (consp tmp1)
                            (progn
                              (setq cond (car tmp1))
                              (setq tmp2 (cdr tmp1))
                              (and (consp tmp2)
                                (eq (cdr tmp2) nil)
                                (progn
                                  (setq value (car tmp2))
                                  t))))))))
                    (t
                      (progn
                        (setq seqTranList (cons item seqTranList))
                        (setq seqList (cdr seqList))))))
                (return (cons 'seq (nreverse seqTranList)))))))
      (if (eql (length seqTranList) 1)
        (car seqTranList)
        (cons 'seq (nreverse seqTranList))))

defun Convert a loop node to an S-expression

TPDHERE: rewrite using dsetq

;loopIters2Sex iterList ==
; result := nil
; for iter in iterList repeat
;   sex := pf2Sex1 iter
;   sex is ['IN, var, ['SEGMENT, i, ['BY", incr]]] =>
;     result := [ ['STEP, var, i, incr], :result]
;   sex is ['IN, var, ['"BY", ['SEGMENT, i, j], incr]] =>
;     result := [ ['STEP, var, i, incr, j], :result]
;   sex is ['IN, var, ['SEGMENT, i, j]] =>
;     result := [ ['STEP, var, i, i, j], :result]
;   result := [sex, :result]
; nreverse result

[pf2Sex1 p530]

— defun loopIters2Sex —

(defun loopIters2Sex (iterList)
  (let (j incr i var sex result tmp1 tmp2 tmp3 tmp4 tmp5 tmp6 tmp7 tmp8)
    (dolist (iter iterList (nreverse result))
      (setq sex (pf2Sex1 iter))
      (cond
        ((and (consp sex)
          (eq (car sex) 'in)
          (progn
            (setq tmp1 (cdr sex))
            (and (consp tmp1)
              (progn
                (setq var (car tmp1))
                (setq tmp2 (cdr tmp1))
                (and (consp tmp2)
                  (eq (cdr tmp2) nil)
                  (progn
                    (setq tmp3 (car tmp2))
                    (and (consp tmp3)
                      (eq (car tmp3) 'segment)
                      (progn
                        (setq tmp4 (cdr tmp3))
                        (and (consp tmp4)
                          (progn
                            (setq i (car tmp4))
                            (setq tmp5 (cdr tmp4))
                            (and (consp tmp5)
                              (eq (cdr tmp5) nil)
                              (progn
                                (setq tmp6 (car tmp5))
                                (and (consp tmp6)
                                  (eq (car tmp6) 'by)
                                  (progn
                                    (setq tmp7 (cdr tmp6))
                                    (and (consp tmp7)
                                      (eq (car tmp7) 'in)
                                      (progn
                                        (setq tmp8 (cdr tmp7))
                                        (and (consp tmp8)
                                          (eq (car tmp8) "; result := nil
"
(eq (cdr tmp7) nil)
(progn
  (setq incr (car tmp7))
  t)))))))))))))))))))

(setq result (cons (list 'step var i incr) result))
((and (consp sex)
  (eq (car sex) 'in)
  (progn
    (setq tmp1 (cdr sex))
    (and (consp tmp1)
      (progn
        (setq var (car tmp1))
        (setq tmp2 (cdr tmp1))
        (and (consp tmp2)
          (eq (cdr tmp2) nil)
          (progn
            (setq tmp3 (car tmp2))
            (and (consp tmp3)
              (eq (car tmp3) 'by)
              (progn
                (setq tmp4 (cdr tmp3))
                (and (consp tmp4)
                  (progn
                    (setq tmp5 (car tmp4))
                    (and (consp tmp5)
                      (eq (car tmp5) 'segment)
                      (progn
                        (setq tmp6 (cdr tmp5))
                        (and (consp tmp6)
                          (progn
                            (setq i (car tmp6))
                            (setq tmp7 (cdr tmp6))
                            (and (consp tmp7)
                              (eq (cdr tmp7) nil)
                              (progn
                                (setq j (car tmp7))
                                t))))))))))
          (progn
            (setq tmp8 (cdr tmp4))
            (and (consp tmp8)
              (eq (cdr tmp8) nil)
              (progn
                (setq incr (car tmp8))
                t))))))))

(setq result (cons (list 'step var i incr) result))
((and (consp sex)
  (eq (car sex) 'in)
  (progn
    (setq tmp1 (cdr sex))
    (and (consp tmp1)
CHAPTER 15. PFTREE TO S-EXPRESSON TRANSLATION

(progn
  (setq var (car tmp1))
  (setq tmp2 (cdr tmp1))
  (and (consp tmp2)
    (eq (cdr tmp2) nil)
    (progn
      (setq tmp3 (car tmp2))
      (and (consp tmp3)
        (eq (car tmp3) 'segment)
        (progn
          (setq tmp4 (cdr tmp3))
          (and (consp tmp4)
            (progn
              (setq i (car tmp4))
              (setq tmp5 (cdr tmp4))
              (and (consp tmp5)
                (eq (cdr tmp5) nil)
                (progn
                  (setq j (car tmp5))
                  t))))))))))))

(setq result (cons (list 'step var i 1 j) result)))
(t (setq result (cons sex result)))))

---

defun Change a Collect node to an S-expression

[loopIters2Sex p541]
[pfParts p479]
[pfCollectIterators p490]
[pf2Sex1 p530]
[pfCollectBody p490]

— defun pfCollect2Sex —

(defun |pfCollect2Sex| (pf)
  (let (var cond sex tmp1 tmp2 tmp3 tmp4)
    (setq sex
      (cons 'collect
        (append ([loopIters2Sex] ([pfParts] ([pfCollectIterators] pf)))
          ([list ([pf2Sex1] ([pfCollectBody] pf))])))
    (cond
      ((and (consp sex)
            (eq (car sex) 'collect)
            (progn
              (setq tmp1 (cdr sex)))
            (and (consp tmp1)
(progn
  (setq tmp2 (car tmp1))
  (and (consp tmp2)
    (eq (car tmp2) '|||
      (progn
        (setq tmp3 (cdr tmp2))
        (and (consp tmp3)
          (eq (cdr tmp3) nil)
          (progn
            (setq cond (car tmp3))
            t))))))
(progn
  (setq tmp4 (cdr tmp1))
  (and (consp tmp4)
    (eq (cdr tmp4) nil)
    (progn (setq var (car tmp4)) t))))
(symbolp var))
(list '||| var cond))
(t sex))))

---

defun Convert a Definition node to an S-expression

[pf2Sex1 p530]
[pf0DefinitionLhsItems p492]
[pfDefinitionRhs p491]
[systemError p??]
[pfLambdaTran p546]
[$insideApplication p??]

--- defun pfDefinition2Sex ---

(defun pfDefinition2Sex (pf)
  (let (body argList tmp1 rhs id idList)
    (declare (special $insideApplication)))
    (if $insideApplication)
      (list 'optarg
        ([pf2Sex1| (car ([pf0DefinitionLhsItems| pf))])
        ([pf2Sex1| ([pfDefinitionRhs| pf]])
        (progn
          (setq idList (mapcar #'|pf2Sex1| ([pf0DefinitionLhsItems| pf])))
          (if (not (eql (length idList) 1))
            (systemError
              "lhs of definition must be a single item in the interpreter")
            (progn
              (setq id (car idList))
              (setq idList 1)) )
          )
    )
  )
defun Convert a Lambda node to an S-expression

(defun pfLambdaTran (pf)
  (let (retType argList argTypeList)
    (cond
      ((pfLambda? pf)
        (dolist (arg (pf0LambdaArgs pf))
          (if (pfTyped? arg)
              (progn
                (setq argList
                  (cons (pfCollectArgTran (pfTypedId arg)) argList))
                (if (pfNothing? (pfTypedType arg))
                    (setq argTypeList (cons nil argTypeList))
                    (setq argTypeList
                      (cons (pf2Sex1 (pfTypedType arg)) argTypeList))))
            (systemError "definition args should be typed")))
      (setq argList (nreverse argList))
      (unless (pfNothing? (pfLambdaRets pf))
        (setq retType (pf2Sex1 (pfLambdaRets pf)))
        (setq argTypeList (cons retType (nreverse argTypeList))))))
(cons argList
 (list argTypeList
   (mapcar #'(lambda (x) (declare (ignore x)) nil) argTypeList)
   (|pf2Sex1| (|pfLambdaBody| pf)))))
(t (cons '|id| (list '(nil) '(nil) (|pf2Sex1| pf))))))

---

defun pfCollectArgTran

defun |pfCollectArgTran| (pf)
(let (cond tmp2 tmp1 id conds)
  (cond
    ((|pfCollect?| pf)
      (setq conds (mapcar #'|pf2sex1| (|pfParts| (|pfCollectIterators| pf))))
      (setq id (|pf2Sex1| (|pfCollectBody| pf)))
      (cond
        ((and (consp conds) ; conds is ["|", cond] ]
          (eq (cdr conds) nil)
          (progn
            (setq tmp1 (car conds))
            (and (consp tmp1)
              (eq (car tmp1) '\|
                (progn
              (setq tmp2 (cdr tmp1))
              (and (consp tmp2)
                (eq (cdr tmp2) nil)
                (progn
              (setq cond (car tmp2))
                t))))))
            (list '\|
              id cond))
          (t (cons id conds))))
      (t (|pf2Sex1| pf))))
  (t (|pf2Sex1| pf))))

---

defun Convert a Lambda node to an S-expression

(defun pfLambda2Sex |pfLambdaTran p546|
  (defun pfLambda2Sex (pf)
    (let (body argList tmp1)
      (setq tmp1 (pfLambdaTran pf))
      (setq argList (car tmp1))
      (setq body (cdr tmp1))
      (cons 'adef (cons argList body))))

defun Convert a Rule node to an S-expression

(defun pfRule2Sex |pfRuleLhsItems p516|
  (defun pfRule2Sex (pf)
    (let ($multiVarPredicateList $predicateList $quotedOpList rhs lhs)
      (declare (special $multiVarPredicateList $predicateList $quotedOpList))
      (setq $quotedOpList nil)
      (setq $predicateList nil)
      (setq $multiVarPredicateList nil)
      (setq lhs (pfLhsRule2Sex (pfRuleLhsItems pf)))
      (setq rhs (pfRhsRule2Sex (pfRuleRhs pf)))
      (setq lhs (ruleLhsTran lhs))
      (rulePredicateTran)
      (if $quotedOpList
        (list '|rule| lhs rhs (cons '|construct| $quotedOpList))
        (list '|rule| lhs rhs))))
defun Convert the Lhs of a Rule to an S-expression

(defvar pf2Sex1 p530)
(defvar $insideRule p??)

--- defun pfLhsRule2Sex ---

(defun pfLhsRule2Sex (lhs)
  (let (($insideRule))
    (declare (special $insideRule))
    (setq $insideRule 'left)
    (pf2Sex1 lhs))

---

defun Convert the Rhs of a Rule to an S-expression

(defvar pf2Sex1 p530)
(defvar $insideRule p??)

--- defun pfRhsRule2Sex ---

(defun pfRhsRule2Sex (rhs)
  (let (($insideRule))
    (declare (special $insideRule))
    (setq $insideRule 'right)
    (pf2Sex1 rhs))

---

defun Convert a Rule predicate to an S-expression

(defvar rulePredicateTran rule =

; null $multiVarPredicateList => rule
; varList := patternVarsOf [rhs for [.,.,:rhs] in $multiVarPredicateList] in $multiVarPredicateList
; predBody :=
; CDR $multiVarPredicateList =>
; ['AND, [pvarPredTran(rhs, varList) for [.,.,:rhs] in $multiVarPredicateList]]
; [.,.,:rhs],:[] := $multiVarPredicateList
; pvarPredTran(rhs, varList)
; ['suchThat, rule,
; ['construct, [:QUOTE, var] for var in varList],
; ['ADEF, '(predicateVariable),
; '((Boolean) (List (Expression (Integer)))), '(() ()
; predBody)]
(defun rulePredicateTran (rule)
  (let (predBody varList rhs tmp1 result)
    (declare (special $multiVarPredicateList))
    (if (null $multiVarPredicateList)
        rule
        (progn
          (setq varList
            (|patternVarsOf|
              ((lambda (t1 t2 t3)
                  (loop
                    (cond
                      ((or (atom t2)
                          (progn
                            (setq t3 (car t2))
                            nil)
                        (return (nreverse t1)))
                      (t
                        (and (consp t3)
                          (progn
                            (setq tmp1 (cdr t3))
                            (and (consp tmp1)
                              (progn
                                (setq rhs (cdr tmp1))
                                t))))
                      (setq t1 (cons rhs t1)))))
            (setq t2 (cdr t2))))
            nil $multiVarPredicateList| nil))
        (setq predBody
          (cond
            ((cdr $multiVarPredicateList))
            (cons 'and
              ((lambda (t4 t5 t6)
                  (loop
                    (cond
                      ((or (atom t5)
                          (progn
                            (setq t6 (car t5))
                            nil))
                        (return (nreverse t4)))
                      (t
                        (and (consp t6)
                          (progn
                            (setq tmp1 (cdr t6))
                            t))))))
            (setq t2 (cdr t2)))))
          nil $multiVarPredicateList| nil))
      (setq predBody
        (cond
          ((cdr $multiVarPredicateList))
          (cons 'and
            ((lambda (t4 t5 t6)
                (loop
                  (cond
                    ((or (atom t5)
                        (progn
                          (setq t6 (car t5))
                          nil))
                    (return (nreverse t4)))
                    (t
                      (and (consp t6)
                        (progn
                          (setq tmp1 (cdr t6))
                          t))))))
          (setq t2 (cdr t2)))))
      nil $multiVarPredicateList| nil))
    (setq result
      (cond
        ((null predBody)
          nil)
        (t
          (cons rhs (cons predBody nil))))))
(and (consp tmp1)
  (progn
    (setq rhs (cdr tmp1))
    t)))
(setq t4
  (append (reverse (pvarPredTran rhs varList))
   t4))))
(setq t5 (cdr t5))))
nil ($multiVarPredicateList| nil)))
(t
  (progn
    (setq rhs (cddar •$multiVarPredicateList|))
    (pvarPredTran rhs varList)))))
(dolist (var varList) (push (list 'quote var) result))
(list 'suchThat| rule
  (cons 'construct| (nreverse result))
  (list 'adef '(|predicateVariable|)
    '(((|Boolean|)
      (|List| (|Expression| (|Integer|)))))
    '((nil nil) predBody)))))

---

defun patternVarsOf
[patternVarsOf1 p551]

    --- defun patternVarsOf ---

(defun patternVarsOf1 (expr)
  (patternVarsOf1 expr nil))

---

defun patternVarsOf1
[patternVarsOf1 p551]

    --- defun patternVarsOf1 ---

(defun patternVarsOf1 (expr varList)
  (let (argl op)
    (cond
      ((null expr) varList)
      ((atom expr)
(cond
  ((null (symbolp expr)) varList)
  ((member expr varList) varList)
  (t (cons expr varList)))
((and (consp expr)
  (progn
    (setq op (car expr))
    (setq argl (cdr expr))
    t))
  (progn
    (dolist (arg argl)
      (setq varList (|patternVarsOf1| arg varList))
    varList))
  (t varList)))

---
defun pvarPredTran

---
defun pvarPredTran —

(defun pvarPredTran (rhs varList)
  (let ((i 0))
    (dolist (var varList rhs)
      (setq rhs (nsubst (list '|elt| '|predicateVariable| (incf i)) var rhs))))

---
defun Convert the Lhs of a Rule node to an S-expression

[patternVarsOf p551]
[nsubst p??]
[$predicateList p??]
[$multiVarPredicateList p??]

---
defun ruleLhsTran —

(defun ruleLhsTran (ruleLhs)
  (let (predicate var vars predRhs predLhs name)
    (declare (special |$predicateList| |$multiVarPredicateList|))
    (dolist (pred |$predicateList|)
      (setq name (car pred))
    (setq predLhs (cadr pred))
    (setq predRhs (cddr pred)))
(setq vars (patternVarsOf predRhs))
(cond
  ((cdr vars)
   (setq ruleLhs (nsubst predLhs name ruleLhs))
   (setq $multiVarPredicateList| (cons pred $multiVarPredicateList|)))
  (t
   (setq var (cadr predLhs))
   (setq predicate
     (list 'suchThat| predLhs (list 'adef (list var)
                      '((|Boolean|) (|Expression| (|Integer|))) '(nil nil) predRhs)))
   (setq ruleLhs (nsubst predicate name ruleLhs)))))
ruleLhs)

---

defun Translate ops into internal symbols

— defun opTran 0 —

(defun |opTran| (op)
  (cond
   ((equal op '|..|) 'segment)
   ((eq op '[]) '|construct|)
   ((eq op '{}) '|braceFromCurly|)
   ((eq op 'is) '|is|)
   (t op)))

---
Chapter 16

Stream Utilities

The input stream is parsed into a large s-expression by repeated calls to Delay. Delay takes a function \( f \) and an argument \( x \) and returns a list consisting of \( (\text{nonnullstream} \ f \ x) \). Eventually multiple calls are made and a large list structure is created that consists of \( (\text{nonnullstream} \ f \ x \ (\text{nonnullstream} \ f_1 \ x_1 \ (\text{nonnullstream} \ f_2 \ x_2...\right) \\
This delay structure is given to StreamNull which walks along the list looking at the head. If the head is “nonnullstream” then the function is applied to the argument.

So, in effect, the input is “zipped up” into a Delay data structure which is then evaluated by calling StreamNull. This "zippered stream" parser was a research project at IBM and Axiom was the testbed (which explains the strange parsing technique).

\[
defun \text{npNull} \\
\text{[StreamNull p555]} \\
\text{— defun npNull —}
\]

\[
defun \text{npNull} \ (x) \ (\text{StreamNull} \ x)) \\
\text{———}
\]

\[
defun \text{StreamNull} \\
\text{[eqcar p??]} \\
\text{StreamNull : Delay → Union(T,NIL)} \\
\text{— defun StreamNull 0 —}
\]

\[
defun \text{StreamNull} \ (\text{delay}) \\
\]

555
(let (parsepair)
  (cond
    ((or (null delay) (eqcar delay '|nullstream|)) t)
    (t
      ((lambda nil
         (loop
           (cond
             ((not (eqcar delay '|nonnullstream|)) (return nil))
             (t
              (setq parsepair (apply (cadr delay) (cddr delay)))
              (rplaca delay (car parsepair))
              (rplacd delay (cdr parsepair)))))
            (eqcar delay '|nullstream|))))
  )
Chapter 17

Code Piles

The insertpiles function converts a line-list to a line-forest where a line is a token-deque and has a column which is an integer. An A-forest is an A-tree-list An A-tree has a root which is an A, and subtrees which is an A-forest.

A forest with more than one tree corresponds to a Scratchpad pile structure (t1;t2;t3;...;tn), and a tree corresponds to a pile item. The ( ; and ) tokens are inserted into a ;1-forest, otherwise the root of the first tree is concatenated with its forest. column t is the number of spaces before the first non-space in line t

defun insertpile

(npNull p555)
madlePlusComment p558

(madlPlusComments p558)

(pileTree p559)
madleCforest p562

— defun insertpile —

(defun insertpile (s)
  (let (stream a t1 h1 t2 h tmp1)
    (cond
     ((npNull s) (list nil 0 nil s))
     (t
      (setq tmp1 (list (car s) (cdr s)))
      (setq h (car tmp1))
      (setq t2 (cadr tmp1))
      (cond
       ((pilePlusComment h)
        (setq tmp1 (pilePlusComments s))
        (setq h1 (car tmp1))
        (setq h1 (car tmp1))
        (setq h1 (car tmp1))
        (setq h1 (car tmp1)))
      (t
       (setq h1 (car tmp1))))
      (setq h1 (car tmp1))))

557
(setq t1 (cadr tmp1))
(setq a (pileTree (- 1) t1))
(cons (list (pileCforest
    (append h1 (cons (elt a 2) nil))))
    (elt a 3)))
(t
(setq stream (cadar s))
(setq a (pileTree -1 s))
(cons (list (list (elt a 2) stream)) (elt a 3))))

(defun pilePlusComment

[tokType p635]
[npNull p555]
[pilePlusComment p558]
[pilePlusComments p558]

— defun pilePlusComment —

(defun |pilePlusComment| (arg)
  (eq (|tokType| (caar arg)) '|comment|))

— defun pilePlusComments —

(defun |pilePlusComments| (s)
  (let (t1 h1 t2 h tmp1)
    (cond
      ((|npNull| s) (list nil s))
    (t
      (setq tmp1 (list (car s) (cdr s)))
      (setq h (car tmp1))
      (setq t2 (cdr tmp1))
      (cond
        ((|pilePlusComment| h)
          (setq tmp1 (|pilePlusComments| t2))
          (setq h1 (car tmp1))
          (setq t1 (cdr tmp1))
          (list (cons h h1) t1))))
(t
  (list nil s)))))))

defun pileTree

[npNull p555]
[pileColumn p559]
[pileForests p559]

    — defun pileTree —

(defun |pileTree| (n s)
  (let ((hh t1 h tmp1)
    (cond
      ((|npNull| s) (list nil n nil s))
      (t
        (setq tmp1 (list (car s) (cdr s)))
        (setq h (car tmp1))
        (setq t1 (cadr tmp1))
        (setq hh (|pileColumn| (car h)))
        (cond
          ((< n hh) (|pileForests| (car h) hh t1))
          (t (list nil n nil s)))))))

    —

(defun pileColumn

[tokPosn p635]

    — defun pileColumn —

(defun |pileColumn| (arg)
  (cdr (|tokPosn| (caar arg))))

    —

(defun pileForests

[pileForest p560]
[npNull p555]
— defun pileForests —

(defun pileForests (h n s)
  (let (t1 h1 tmp1)
    (setq tmp1 (pileForest n s))
    (setq h1 (car tmp1))
    (setq t1 (cadr tmp1))
    (cond
      ((null h1) (list t n h s))
      (t (pileForests (pileCtree h h1) n t1)))))

defun pileForest

— defun pileForest —

(defun pileForest (n s)
  (let (t1 h1 t2 h hh b tmp)
    (setq tmp (pileTree n s))
    (setq b (car tmp))
    (setq hh (cadr tmp))
    (setq h (caddr tmp))
    (setq t2 (cadddr tmp))
    (cond
      (b
        (setq tmp (pileForest1 hh t2))
        (setq h1 (car tmp))
        (setq t1 (cadr tmp))
        (list (cons h h1) t1))
      (t
        (list nil s)))))

defun pileForest1

— defun pileForest1 —

(defun pileForest1 (n s)
  (let (t1 h1 t2 h hh b tmp)
    (setq tmp (pileTree n s))
    (setq b (car tmp))
    (setq hh (cadr tmp))
    (setq h (caddr tmp))
    (setq t2 (cadddr tmp))
    (cond
      (b
        (setq tmp (pileForest1 hh t2))
        (setq h1 (car tmp))
        (setq t1 (cadr tmp))
        (list (cons h h1) t1))
      (t
        (list nil s)))))
--- defun pileForest1 ---

(defun pileForest1 (n s)
  (let (t1 h1 t2 h n1 b tmp)
    (setq tmp (eqpileTree n s))
    (setq b (car tmp))
    (setq n1 (cadr tmp))
    (setq h (caddr tmp))
    (setq t2 (cadddr tmp))
    (cond
      (b
       (setq tmp (pileForest1 n t2))
       (setq h1 (car tmp))
       (setq t1 (cadr tmp))
       (list (cons h h1) t1))
      (t (list nil s))))

---

defun eqpileTree

[npNull p555]
[pileColumn p559]
[pileForests p559]

--- defun eqpileTree ---

(defun eqpileTree (n s)
  (let (hh t1 h tmp)
    (cond
      ((npNull s) (list nil n nil s))
      (t
       (setq tmp (list (car s) (cdr s)))
       (setq h (car tmp))
       (setq t1 (cadr tmp))
       (setq hh (pileColumn (car h)))
       (cond
        ((equal hh n) (pileForests (car h) hh t1))
        (t (list nil n nil s)))))))
defun pileCtree

[dqAppend p566]
[pileCforest p562]

— defun pileCtree —

(defun pileCtree (x y)
  (dqAppend x (pileCforest y)))

—

defun pileCforest

Only enpiles forests with >= 2 trees
[tokPart p635]
[enPile p562]
[separatePiles p563]

— defun pileCforest —

(defun pileCforest (x)
  (let (f)
    (cond
      ((null x) nil)
      ((null (cdr x)) (setq f (car x))
        (cond
          ((eq (tokPart (caar f)) 'if) (enPile f))
          (t f))
        (t (enPile (separatePiles x)))))))

—

defun enPile

[dqConcat p565]
[dqUnit p565]
[tokConstruct p633]
[firstTokPosn p563]
[lastTokPosn p563]

— defun enPile —

(defun enPile (x)
(list
   (conc (list (const 'key) (const 'settab (list (var x)))
          (list (const 'key) (const 'backtab (list (var x)))))))

---

defun firstTokPosn

(tokPosn p635)

   --- defun firstTokPosn ---

   (defun firstTokPosn (arg) (list (var (caar arg))))

---

defun lastTokPosn

(tokPosn p635)

   --- defun lastTokPosn ---

   (defun lastTokPosn (arg) (list (var (cadr arg))))

---

defun separatePiles

(DqUnit p565)
(tokConstruct p633)
(lastTokPosn p563)
(DqConcat p565)
(separatePiles p563)

   --- defun separatePiles ---

   (defun separatePiles (x)
      (let (semicolon a)
         (cond
            ((null x) nil)
            ...)))
((null (cdr x)) (car x))
(t
(setq a (car x))
(setq semicolon
  (|dqUnit| (|tokConstruct| 'key 'backset (|lastTokPosn| a)))
  (|dqConcat| (list a semicolon (|separatePiles| (cdr x)))))))
Chapter 18

Deque Functions

The dqUnit makes a unit dq i.e. a dq with one item, from the item

```
defun dqUnit
--- defun dqUnit 0 ---

(defun |dqUnit| (s)
  (let (a)
    (setq a (list s))
    (cons a a)))
```

---

```
defun dqConcat

The dqConcat function concatenates a list of dq’s, destroying all but the last
[dqAppend p566]
dqConcat p565

--- defun dqConcat ---

(defun |dqConcat| (ld)
  (cond
    ((null ld) nil)
    ((null (cdr ld)) (car ld))
    (t (|dqAppend| (car ld) (|dqConcat| (cdr ld)))))))
```
defun dqAppend

The dqAppend function appends 2 dq's, destroying the first

(defun dqAppend (x y)
  (cond
   ((null x) y)
   ((null y) x)
   (t
    (rplacd (cdr x) (car y))
    (rplacd x (cdr y)) x)))

defun dqToList

(defun dqToList (s)
  (when s (car s)))
Chapter 19

Message Handling

19.1 The Line Object

defun Line object creation

This is called in only one place, the incLine1 function.

— defun lnCreate 0 —

(defun lnCreate (extraBlanks string globalNum &rest optFileStuff)
  (let ((localNum (first optFileStuff))
          (filename (second optFileStuff)))
    (unless localNum (setq localNum 0))
    (list extraBlanks string globalNum localNum filename)))

—

defun Line element 0; Extra blanks

— defun lnExtraBlanks 0 —

(defun lnExtraBlanks (lineObject) (elt lineObject 0))

—

defun Line element 1; String

— defun lnString 0 —

567
(defun lnString (lineObject) (elt lineObject 1))

---

defun Line element 2; Global number

— defun lnGlobalNum 0 —

(defun lnGlobalNum (lineObject) (elt lineObject 2))

---

defun Line element 2; Set Global number

— defun lnSetGlobalNum 0 —

(defun lnSetGlobalNum (lineObject num)
  (setf (elt lineObject 2) num))

---

defun Line element 3; Local number

— defun lnLocalNum 0 —

(defun lnLocalNum (lineObject) (elt lineObject 3))

---

defun Line element 4; Place of origin

— defun lnPlaceOfOrigin 0 —

(defun lnPlaceOfOrigin (lineObject) (elt lineObject 4))

---
defun Line element 4: Is it a filename?

— defun lnImmediate? 0 —

(defun |lnImmediate?| (lineObject) (null (|lnFileName?| lineObject)))

defun Line element 4: Is it a filename?

— defun lnFileName? 0 —

(defun |lnFileName?| (lineObject)
  (let (filename)
    (when (consp (setq filename (elt lineObject 4))) filename)))

defun Line element 4; Get filename

— defun lnFileName —

(defun |lnFileName| (lineObject)
  (let (fN)
    (if (setq fN (|lnFileName?| lineObject))
      fN
      (|ncBug| "there is no file name in %1" (list lineObject)))))

19.2 Messages

defun msgCreate

msgObject
  tag -- catagory of msg
-- attributes as a-list
'impR => dont save for list processing
toWhere, screen or file
'norep => only display once in list
pos -- position with possible FROM/TO tag
key -- key for message database
argL -- arguments to be placed in the msg test
prefix -- things like "Error: "
text -- the actual text

[setMsgForcedAttrList p586]
[putDatabaseStuff p587]
[initImPr p589]
[initToWhere p590]

— defun msgCreate —

(defun |msgCreate| (tag posWTag key argL optPre &rest optAttr)
  (let (msg)
    (when (consp key) (setq tag '|old|))
    (setq msg (list tag posWTag key argL optPre nil))
    (when (car optAttr) (|setMsgForcedAttrList| msg (car optAttr)))
    (|putDatabaseStuff| msg)
    (|initImPr| msg)
    (|initToWhere| msg)
    msg))

defmacro getMsgPosTagOb

— defmacro getMsgPosTagOb 0 —

(defun |getMsgPosTagOb| (msg)
  '(elt ,msg 1))

— defmacro getMsgKey —

defmacro getMsgKey

— defmacro getMsgKey 0 —
(defmacro |getMsgKey| (msg)
  `(elt ,msg 2))

---

defmacro getMsgArgL

---

defmacro getMsgArgL 0

(defmacro |getMsgArgL| (msg)
  `(elt ,msg 3))

---

defmacro getMsgPrefix

---

defmacro getMsgPrefix 0

(defmacro |getMsgPrefix| (msg)
  `(elt ,msg 4))

---

defmacro setMsgPrefix

---

defmacro setMsgPrefix 0

(defmacro |setMsgPrefix| (msg val)
  `(setf (elt ,msg 4),val))

---

defmacro getMsgText

---

defmacro getMsgText 0
(defmacro |getMsgText| (msg)
  '(elt ,msg 5))

defmacro setMsgText

— defmacro setMsgText 0 —

(defmacro |setMsgText| (msg val)
  '(setf (elt ,msg 5) ,val))

—

defmacro getMsgPrefix?

— defmacro getMsgPrefix? 0 —

(defmacro |getMsgPrefix?| (msg)
  '(let ((pre (|getMsgPrefix| ,msg)))
    (unless (eq pre '|noPre|) pre)))

—

defmacro getMsgTag

The valid message tags are: line, old, error, warn, bug, unimple, remark, stat, say, debug [ncTag p637]

— defmacro getMsgTag 0 —

(defmacro |getMsgTag| (msg)
  '(|ncTag| ,msg))

—

defmacro getMsgTag?

[ifcar p??]
[getMsgTag p572]
19.2. MESSAGES

— defmacro getMsgTag? 0 —

(defmacro |getMsgTag?| (msg)
  `(ifcar (member (|getMsgTag| ,msg)
    (list '|line| '|old| '|error| '|warn| '|bug|
      '|unimple| '|remark| '|stat| '|say| '|debug|))))

———

defmacro line?

[getMsgTag p572]

— defmacro line? —

(defmacro |line?| (msg)
  `(eq (|getMsgTag| ,msg) '|line|))

———

defmacro leader?

[getMsgTag p572]

— defmacro leader? —

(defmacro |leader?| (msg)
  `(eq (|getMsgTag| ,msg) '|leader|))

———

defmacro toScreen?

[getMsgToWhere p585]

— defmacro toScreen? —

(defmacro |toScreen?| (msg)
  `(not (eq (|getMsgToWhere| ,msg) '|fileOnly|)))

———
defun ncSoftError

Messages for the USERS of the compiler. The program being compiled has a minor error. Give a message and continue processing.

[desiredMsg p575]
[processKeyedError p575]
[msgCreate p569]
[$newcompErrorCount p251]

— defun ncSoftError —

(defun ncSoftError (pos erMsgKey erArgL &rest optAttr)
  (declare (special $newcompErrorCount))
  (setq $newcompErrorCount (+ $newcompErrorCount 1))
  (when (desiredMsg erMsgKey)
    (processKeyedError
     (msgCreate 'error pos erMsgKey erArgL
        "Error" optAttr))))

—

defun ncHardError

The program being compiled is seriously incorrect. Give message and throw to a recovery point.

[desiredMsg p575]
[processKeyedError p575]
[msgCreate p569]
[ncError p293]
[$newcompErrorCount p251]

— defun ncHardError —

(defun ncHardError (pos erMsgKey erArgL &rest optAttr)
  (declare (special $newcompErrorCount))
  (setq $newcompErrorCount (+ $newcompErrorCount 1))
  (if (desiredMsg erMsgKey)
       (processKeyedError
        (msgCreate 'error pos erMsgKey erArgL
           "Error" optAttr))
       (ncError)))

—
defun desiredMsg

--- defun desiredMsg 0 ---

(defun |desiredMsg| (erMsgKey &rest optCatFlag)
 (declare (ignore erMsgKey))
 (cond
   ((null (null optCatFlag)) (car optCatFlag))
   (t t)))

defun processKeyedError

(defun |processKeyedError| (msg)
 (prog (pre erMsg)
 (declare (special |$ncMsgList|))
 (cond
   ((eq (|getMsgTag?| msg) '|old|)
    (setq erMsg (|getMsgKey| msg))
    (cond
     ((setq pre (|getMsgPrefix?| msg))
      (setq erMsg (cons pre erMsg)))))
   (|sayBrightly| (cons "old msg from " (cons (|CallerName| 4) erMsg)))
   ((|msgImPr?| msg) (|msgOutputter| msg))
   (t (setq |$ncMsgList| (cons msg |$ncMsgList|))))

---

defun msgOutputter

(defun |msgOutputter| (msg)
 (getStFromMsg msg)
 (leader?))
CHAPTER 19. MESSAGE HANDLING

(defun msgOutputter | msg |
(let (alreadyOpened shouldFlow st)
  (declare (special $linelength))
  (setq st (getStFromMsg | msg |))
  (setq shouldFlow (null (or (leader? | msg |) (line? | msg |))))
  (when (toScreen? | msg |)
    (when shouldFlow (setq st (flowSegmentedMsg | st | $linelength 0)))
    (sayBrightly | st |))
  (when (toFile? | msg |)
    (when shouldFlow (setq st (flowSegmentedMsg | st | (- $linelength 6) 0)))
    (setq alreadyOpened (alreadyOpened? | msg |))))

(defun listOutputter | outputList |
(defun listOutputter | outputList |
  (dolist (msg outputList)
    (msgOutputter | msg |)))

defun getStFromMsg

(defun getStFromMsg | msg |
  (getPreStL | msg |)
  (getMsgPrefix? | msg |)
  (getMsgTag | msg |)
  (getMsgText | msg |)
  (getPosStL | msg |)
  (getMsgKey? | msg |))
19.2. MESSAGES

<table>
<thead>
<tr>
<th>defun getStFromMsg</th>
</tr>
</thead>
</table>
| (defun |getStFromMsg| (msg)
| (let (st posStL preStL)
| (setq preStL (|getPreStL| (|getMsgPrefix?| msg)))
| (cond
| ((eq (|getMsgTag| msg) '|line|)
| (cons "" (append preStL (cons (|getMsgText| msg) nil))))))
| (t
| (setq posStL (|getPosStL| msg))
| (setq st
| (cons posStL
| (cons " "
| (append preStL
| (cons (|tabbing| msg)
| (|getMsgText| msg)))))
| ))))))
| | |

defvar $preLength

— initvars —

(defvar |$preLength| 11)

---

defun getPreStL

| size p1196 |
| $preLength p577 |

— defun getPreStL 0 —

(defun |getPreStL| (optPre)
| (let (apse extraPlaces)
| (declare (special |$preLength|))
| (cond
| ((null optPre) (list " "))
| (t
(setq spses
  (cond
    ((< 0 (setq extraPlaces (- (- |$preLength| (size optPre)) 3)))
      (make-string extraPlaces))
    (t ""))
  (list optPre spses ":"))))

defun getPosStL
          (defun getPosStL (msg)
            (let (printedOrigin printedLineNum printedFileName fullPrintedPos howMuch
                  msgPos)
              (declare (special |$lastPos|))
              (cond
                ((null (|showMsgPos?| msg)) "")
                (t
                  (setq msgPos (|getMsgPos| msg))
                  (setq howMuch
                    (if (|msgImPr?| msg)
                        (|decideHowMuch| msgPos |$lastPos|)
                        (|listDecideHowMuch| msgPos |$lastPos|)))
                  (setq |$lastPos| msgPos)
                  (setq fullPrintedPos (|ppos| msgPos))
                  (setq printedFileName
                    (cons "%x2" (cons "[" (append (|remFile| fullPrintedPos) (cons "]" nil)))))
                  (setq printedLineNum
                    (cons "%x2" (cons "[" (append (|remLine| fullPrintedPos) (cons "]" nil)))))
                  (setq printedOrigin
                    (cons "%x2" (cons "[" (append fullPrintedPos (cons "]" nil)))))
                  (cond
                    ((eq howMuch 'org)
                      (cons "" (append printedOrigin (cons "|%l| nil)))))
                    ((eq howMuch 'line)
                      (cons "" (append printedLineNum (cons "|%l| nil)))))
                    ((eq howMuch 'file)
                      (cons "" (append printedFileName (cons "|%l| nil)))))
                    (t nil)))
        (setq spses
          (cond
            ((< 0 (setq extraPlaces (- (- |$preLength| (size optPre)) 3)))
              (make-string extraPlaces))
            (t ""))
          (list optPre spses ":"))))

— defun getPosStL —

(cons "" (append printedLineNum (cons '|%l| nil))))
((eq howMuch 'file)
 (cons "" (append printedFileName (cons '|%l| nil))))
((eq howMuch 'all)
 (cons ""
 (append printedFileName
 (cons '|%l|
 (cons ""
 (append printedLineNum
 (cons '|%l| nil))))))))
(t ""))))

---

defun ppos

[|pfNoPosition| p634]
[|pfImmediate| p?]
[|pfCharPosn| p465]
[|pfLinePosn| p465]
[|porigin| p313]
[|pfFileName| p466]

---

defun ppos

(let (org lpos cpos)
 (cond
 (((|pfNoPosition|) p) (list "no position"))
 (((|pfImmediate|) p) (list "console"))
 (t
 (setq cpos (|pfCharPosn| p))
 (setq lpos (|pfLinePosn| p))
 (setq org (|porigin| (|pfFileName| p)))
 (list org " " "line" " " lpos))))

---

defun remFile

[|ifcdr| p?]
[|ifcar| p?]

---

defun remFile
(defun remFile (positionList) (ifcdr (ifcdr positionList)))

---

defun showMsgPos?

[msgImPr? p580]
[leader? p573]
[$erMsgToss p??]

— defun showMsgPos? 0 —

(defun showMsgPos? (msg)
  (declare (special $erMsgToss))
  (or $erMsgToss (and (null (msgImPr? msg)) (null (leader? msg)))))

---

defvar $imPrGuys

— initvars —

(defun $imPrGuys (list 'imPr))

---

defun msgImPr?

[getMsgCatAttr p580]

— defun msgImPr? —

(defun msgImPr? (msg)
  (eq (getMsgCatAttr msg '$imPrGuys) 'imPr))

---

defun getMsgCatAttr

[ifcdr p??]
[qassq p??]
19.2. MESSAGES

[ncAlist p637]

— defun getMsgCatAttr —

(defun getMsgCatAttr (msg cat)
  (ifcdr (qassq cat (incAlist msg))))

———

defun getMsgPos

[getMsgFTTag? p581]
[getMsgPosTagOb p570]

— defun getMsgPos —

(defun getMsgPos (msg)
  (if (getMsgFTTag? msg)
      (cadr (getMsgPosTagOb msg))
      (getMsgPosTagOb msg)))

———

defun getMsgFTTag?

[ifcar p537]
[getMsgPosTagOb p570]

— defun getMsgFTTag? —

(defun getMsgFTTag? (msg)
  (ifcar (member (ifcar (getMsgPosTagOb msg)) (list 'from 'to 'fromto)))))

———

defun decideHowMuch

When printing a msg, we wish not to show pos information that was shown for a previous
msg with identical pos info. org prints out the word noposition or console [poNopos? p582]
[poPosImmediate? p582]
[poFileName p582]
[poLinePosn p583]
— defun decideHowMuch —

(defun decideHowMuch (pos oldPos)
  (cond
    ((or (and (poNopos? pos) (poNopos? oldPos))
         (and (poPosImmediate? pos) (poPosImmediate? oldPos)))
      'none)
    ((or (poNopos? pos) (poPosImmediate? pos)) 'org)
    ((or (poNopos? oldPos) (poPosImmediate? oldPos)) 'all)
    ((not (equal (poFileName oldPos) (poFileName pos))) 'all)
    ((not (equal (poLinePosn oldPos) (poLinePosn pos))) 'line)
    (t 'none)))

defun poNopos?

— defun poNopos? 0 —

(defun poNopos? (posn)
  (equal posn (list 'noposition)))

defun poPosImmediate?

[poNopos? p582]
[lnImmediate? p569]
[poGetLineObject p583]

— defun poPosImmediate? —

(defun poPosImmediate? (txp)
  (unless (poNopos? txp) (lnImmediate? (poGetLineObject txp))))

defun poFileName

[lnFileName p569]
[poGetLineObject p583]
--- defun poFileName ---

(defun poFileName (posn)
  (if posn
      (lnFileName (poGetLineObject posn))
      (caar posn)))

---

defun poGetLineObject

defun poGetLineObject 0

(defun poGetLineObject (posn)
  (car posn))

---

defun poLinePosn

[lnLocalNum p568]
[poGetLineObject p583]

madf defun poLinePosn ---

(defun poLinePosn (posn)
  (if posn
      (lnLocalNum (poGetLineObject posn))
      (cdar posn)))

---

defun listDecideHowMuch

[poNopos? p582]
[poPosImmediate? p582]
[poGlobalLinePosn p296]

--- defun listDecideHowMuch ---
(defun listDecideHowMuch (pos oldPos)
  (cond
   ((or (and (poNopos? pos) (poNopos? oldPos))
       (and (poPosImmediate? pos) (poPosImmediate? oldPos)))
    'none)
   ((poNopos? pos) 'org)
   ((poNopos? oldPos) 'none)
   ((< (poGlobalLinePosn pos) (poGlobalLinePosn oldPos))
    (if (poPosImmediate? pos) 'org 'line))
   (t 'none)))

_____

defun remLine

__— defun remLine 0 —__

(defun remLine (positionList) (list (ifcar positionList)))

_____

defun getMsgKey?

[identp p1197]

__— defun getMsgKey? 0 —__

(defun getMsgKey? (msg)
  (let ((val (getMsgKey msg)))
    (when (identp val) val)))

_____

defun tabbing

[getMsgPrefix? p572]
[&preLength p577]

__— defun tabbing —__

(defun tabbing (msg)
  (let (chPos)
19.2. MESSAGES

```lisp
(declare (special |$preLength|))
(setq chPos 2)
(when (|getMsgPrefix?| msg) (setq chPos (- (+ chPos |$preLength|) 1)))
(cons '|%t| chPos)))

defvar $toWhereGuys

— initvars —

(defvar |$toWhereGuys| (list '|fileOnly| '|screenOnly|))

defun getMsgToWhere

[getMsgCatAttr p580]

— defun getMsgToWhere —

(defun |getMsgToWhere| (msg) (|getMsgCatAttr| msg |$toWhereGuys|))

defun toFile?

[toFile? p585]

[|fn p??|]

— defun toFile? —

(defun |toFile?| (msg)
  (and (not (eq (|getMsgToWhere| msg) '|screenOnly|))))

defun alreadyOpened?

[|msgImPr?| p580]

— defun alreadyOpened? —
(defun |alreadyOpened?| (msg) (null (|msgImPr?| msg)))

---

defun setMsgForcedAttrList

[setMsgForcedAttr p586]
[whichCat p587]

     — defun setMsgForcedAttrList —

(defun |setMsgForcedAttrList| (msg attrlist)
  (dolist (attr attrlist)
    (|setMsgForcedAttr| msg (|whichCat| attr) attr)))

---

defun setMsgForcedAttr

[setMsgCatlessAttr p587]
[ncPutQ p638]

     — defun setMsgForcedAttr —

(defun |setMsgForcedAttr| (msg cat attr)
  (if (eq cat '|catless|)
      (|setMsgCatlessAttr| msg attr)
      (|ncPutQ| msg cat attr)))

---

defvar $attrCats

     — initvars —

(defvar|$attrCats| (list '|$imPrGuys| '|$toWhereGuys| '|$repGuys|))

---
defun whichCat

[ListMember? p?]
[\$attrCats p586]

--- defun whichCat ---

(defun whichCat (attr)
  (let ((found 'catless) done)
    (declare (special \$attrCats))
    (loop for cat in \$attrCats do
      (when ([ListMember?] attr (eval cat))
        (setq found cat)
        (setq done t)
      )
    until done)
    found))

---

defun setMsgCatlessAttr

TPDHERE: Changed from —catless— to ’—catless—

[ncPutQ p638]
[ifcdr p??]
[qassq p??]
[ncAlist p637]

--- defun setMsgCatlessAttr ---

(defun setMsgCatlessAttr (msg attr)
  (\ncPutQ\ msg catless (cons attr (ifcdr (qassq catless (\ncAlist\ msg))))))

---

defun putDatabaseStuff

TPDHERE: The variable al is undefined [getMsgInfoFromKey p588]
[setMsgUnforcedAttrList p588]
[setMsgText p572]

--- defun putDatabaseStuff ---

(defun putDatabaseStuff (msg)
  (let (attributes text tmp)
(setq tmp (|getMsgInfoFromKey| msg))
(setq text (car tmp))
(setq attributes (cadr tmp))
(when attributes (|setMsgUnforcedAttrList| msg attributes))
(|setMsgText| msg text)))

---

defun getMsgInfoFromKey

[getMsgKey? p584]
[getErFromDbL p72]
[getMsgKey p570]
[segmentKeyedMsg p28]
[removeAttributes p72]
[substituteSegmentedMsg p72]
[getMsgArgL p571]
[$msgDatabaseName p131]

— defun getMsgInfoFromKey —

(defun |getMsgInfoFromKey| (msg)
  (let ([|$msgDatabaseName| attributes tmp msgText msgKey])
    (declare (special |$msgDatabaseName|)))
    (setq |$msgDatabaseName| nil)
    (setq msgText
      (cond
        ((setq msgKey (|getMsgKey?| msg))
         msgKey)
        (t (|getMsgKey| msg)))
      (setq msgText (|segmentKeyedMsg| msgText))
      (setq tmp (|removeAttributes| msgText))
      (setq msgText (car tmp))
      (setq attributes (cadr tmp))
      (setq msgText (|substituteSegmentedMsg| msgText (|getMsgArgL| msg)))
      (list msgText attributes)))

---

defun setMsgUnforcedAttrList

[setMsgUnforcedAttr p589]
[whichCat p587]

— defun setMsgUnformedAttrList —
19.2. MESSAGES

(defun |setMsgUnforcedAttrList| (msg attrlist)
  (dolist (attr attrlist)
    (|setMsgUnforcedAttr| msg (|whichCat| attr) attr)))

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>defun setMsgUnforcedAttr</td>
<td></td>
</tr>
</tbody>
</table>

(setMsgCatlessAttr p587)
[qassq p??]
[ncAlist p637]
[ncPutQ p638]

— defun setMsgUnforcedAttr —

(defun |setMsgUnforcedAttr| (msg cat attr)
  (cond
   ((eq cat '|catless|) (|setMsgCatlessAttr| msg attr))
   ((null (qassq cat (|ncAlist| msg))) (|ncPutQ| msg cat attr))))

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>defvar $imPrTagGuys</td>
<td></td>
</tr>
</tbody>
</table>

| initvars |

(defvar |$imPrTagGuys| (list '|unimple| '|bug| '|debug| '|say| '|warn|))

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>defun initImPr</td>
<td></td>
</tr>
</tbody>
</table>

[getMsgTag p572]
[setMsgUnforcedAttr p589]
|$imPrTagGuys p589|
[$erMsgToss p??]

— defun initImPr —

(defun |initImPr| (msg)
  (declare (special |$imPrTagGuys| |$erMsgToss|)))
(when (or |$erMsgToss| (member (|getMsgTag| msg) |$imPrTagGuys|))
  (|setMsgUnforcedAttr| msg '|$imPrGuys| '|imPr|)))

defun initToWhere

[getMsgCatAttr p580]
[setMsgUnforcedAttr p589]

— defun initToWhere —

(defun initToWhere (msg)
  (if (member '|trace| (|getMsgCatAttr| msg '|catless|))
      (|setMsgUnforcedAttr| msg '|$toWhereGuys| '|screenOnly|)))

defun Report a bug in the compiler

Bug in the compiler: something which shouldn’t have happened did.

[processKeyedError p575]
[msgCreate p569]
[enable-backtrace p??]
[ncAbort p??]
[$nopos p251]
[$newcompErrorCount p251]

— defun ncBug —

(defun ncBug (erMsgKey erArgL &rest optAttr)
  (let (erMsg)
    (declare (special |$nopos| |$newcompErrorCount|))
    (setq |$newcompErrorCount| (+ |$newcompErrorCount| 1))
    (setq erMsg
      (|processKeyedError|
        (|msgCreate| '|bug| |$nopos| erMsgKey erArgL "Bug!" optAttr)))
    (break)
    (|ncAbort|)))

———
defun processMsgList

[erMsgSort p591]
[makeMsgFromLine p593]
[poGlobalLinePosn p206]
[getMsgPos p581]
[queueUpErrors p594]
[listOutputter p576]
[$noRepList p?]
[$outputList p?]

— defun processMsgList —

(defun |processMsgList| (erMsgList lineList)
  (let ([|$noRepList| |$outputList| st globalNumOfLine msgLine])
    (declare (special |$noRepList| |$outputList|))
    (setq |$outputList| nil)
    (setq |$noRepList| nil)
    (setq erMsgList (|erMsgSort| erMsgList))
    (dolist (line lineList)
      (setq msgLine (|makeMsgFromLine| line))
      (setq |$outputList| (cons msgLine |$outputList|))
      (setq globalNumOfLine (|poGlobalLinePosn| (|getMsgPos| msgLine)))
      (setq erMsgList (|queueUpErrors| globalNumOfLine erMsgList)))
    (setq |$outputList| (append erMsgList |$outputList|))
    (setq st "---------SOURCE-TEXT-&-ERRORS------------------------")
    (|listOutputter| (reverse |$outputList|))))

———

defun erMsgSort

[erMsgSep p592]
[listSort p?]

— defun erMsgSort —

(defun |erMsgSort| (erMsgList)
  (let (msgWOPos msgWPos tmp)
    (setq tmp (|erMsgSep| erMsgList))
    (setq msgWPos (car tmp))
    (setq msgWOPos (cadr tmp))
    (setq msgWPos (|listSort| '#|erMsgCompare| msgWPos))
    (setq msgWOPos (reverse msgWOPos))
    (append msgWOPos msgWPos)))
defun erMsgCompare

(defvar compareposns p592)
(defvar getMsgPos p581)

— defun erMsgCompare —

(defun erMsgCompare (ob1 ob2)
  (compareposns (getMsgPos ob2) (getMsgPos ob1)))

defun compareposns

(defvar poGlobalLinePosn p296)
(defvar poCharPosn p599)

— defun compareposns —

(defun compareposns (a b)
  (let ((c d))
    (setq c (poGlobalLinePosn a))
    (setq d (poGlobalLinePosn b))
    (if (equal c d)
      (not (< (poCharPosn a) (poCharPosn b)))
      (not (< c d)))))

defun erMsgSep

(defvar poNopos? p582)
(defvar getMsgPos p581)

— defun erMsgSep —

(defun erMsgSep (erMsgList)
  (let (msgWOPos msgWPos)
    (dolist (msg erMsgList)
      (if (poNopos? (getMsgPos msg))
        (setq msgWOPos (cons msg msgWOPos)))
      (setq msgWPos (cons msg msgWPos))))
\( \text{(setq msgWPos (cons msg msgWPos))} \)
\( \text{(list msgWPos msgWOPos))} \)

---

\textbf{defun makeMsgFromLine}

\texttt{[getlinepos p594]}
\texttt{getlineText p594}
\texttt{[poGlobalLinePosn p296]}
\texttt{[poLinePosn p583]}
\texttt{[strconc p??]}
\texttt{[rep p593]}
\texttt{[char p??]}
\texttt{[size p1196]}
\texttt{[\$preLength p577]}

---

\textbf{— defun makeMsgFromLine —}

\begin{verbatim}
(defun makeMsgFromLine (line)
  (let (localNumOfLine stNum globalNumOfLine textOfLine posOfLine)
    (declare (special \$preLength))
    (setq posOfLine (getlinePos line))
    (setq textOfLine (getlineText line))
    (setq globalNumOfLine (poGlobalLinePosn posOfLine))
    (setq stNum (princ-to-string (poLinePosn posOfLine)))
    (setq localNumOfLine
      (strconc (rep \#space (- \$preLength 7 (size stNum))) stNum))
    (list \line posOfLine nil nil (strconc "Line" localNumOfLine) textOfLine)))
\end{verbatim}

---

\textbf{defun rep}

\texttt{TPDHERE: This function should be replaced by fillerspaces}

---

\textbf{— defun rep 0 —}

\begin{verbatim}
(defun rep (c n)
  (if (< 0 n)
    (make-string n :initial-element (character c))
    "")
\end{verbatim}

---
defun getLinePos

    — defun getLinePos 0 —

(defun |getLinePos| (line) (car line))

-----

defun getLineText

    — defun getLineText 0 —

(defun |getLineText| (line) (cdr line))

-----

defun queueUpErrors

;queueUpErrors(globalNumOfLine,msgList)==
;  thisPosMsgs := []
;  notThisLineMsgs := []
;  for msg in msgList _
;    while thisPosIsLess(getMsgPos msg,globalNumOfLine) repeat
;      --these are msgs that refer to positions from earlier compilations
;      if not redundant (msg,notThisPosMsgs) then
;        notThisPosMsgs := [msg,:notThisPosMsgs]
;      msgList := rest msgList
;    for msg in msgList _
;      while thisPosIsEqual(getMsgPos msg,globalNumOfLine) repeat
;        if not redundant (msg,thisPosMsgs) then
;          thisPosMsgs := [msg,:thisPosMsgs]
;        msgList := rest msgList
;      if thisPosMsgs then
;        thisPosMsgs := processChPosesForOneLine  thisPosMsgs
;        $outputList := NCONC(thisPosMsgs,$outputList)
;      if notThisPosMsgs then
;        $outputList := NCONC(notThisPosMsgs,$outputList)
;      msgList

[processChPosesForOneLine p598]
[$outputList p??]

    — defun queueUpErrors —
(DEFUN queueUpErrors (globalNumOfLine msgList)
 (PROG (notThisPosMsgs notThisLineMsgs thisPosMsgs)
   (DECLARE (SPECIAL $outputList))
   (RETURN
    (PROGN
     (SETQ thisPosMsgs NIL)
     (SETQ notThisLineMsgs NIL)
     ((LAMBDA (bfVar#7 msg)
       (LOOP
        (COND
         ((OR (ATOM bfVar#7)
            (PROGN (SETQ msg (CAR bfVar#7)) NIL)
            (NOT (thisPosIsLess (getMsgPos msg) globalNumOfLine)))
          (RETURN NIL))
         ('T
          (PROGN
           (COND
            ((NULL (redundant msg notThisPosMsgs))
             (SETQ notThisPosMsgs)
             (CONS msg notThisPosMsgs))
            (SETQ msgList (CDR msgList))))
         (SETQ bfVar#7 (CDR bfVar#7))))))
     (SETQ msgList NIL)
     ((LAMBDA (bfVar#8 msg)
       (LOOP
        (COND
         ((OR (ATOM bfVar#8))
          (PROGN (SETQ msg (CAR bfVar#8)) NIL)
          (NOT (thisPosIsEqual (getMsgPos msg) globalNumOfLine)))
          (RETURN NIL))
         ('T
          (PROGN
           (COND
            ((NULL (redundant msg thisPosMsgs))
             (SETQ thisPosMsgs)
             (CONS msg thisPosMsgs))
            (SETQ msgList (CDR msgList))))
         (SETQ bfVar#8 (CDR bfVar#8))))))
     (COND
      (thisPosMsgs)
      (SETQ thisPosMsgs)
      (processChPosesForOneLine thisPosMsgs)))
     (SETQ $outputList (NCONC thisPosMsgs $outputList)))
     (COND
      (notThisPosMsgs)
      (SETQ $outputList (NCONC notThisPosMsgs $outputList))))
   (msgList))))}
defun thisPosIsLess
[poNopos? p582]
[poGlobalLinePosn p296]

— defun thisPosIsLess —

(defun |thisPosIsLess| (pos num)
  (unless (|poNopos?| pos) (< (|poGlobalLinePosn| pos) num)))

defun thisPosIsEqual
[poNopos? p582]
[poGlobalLinePosn p296]

— defun thisPosIsEqual —

(defun |thisPosIsEqual| (pos num)
  (unless (|poNopos?| pos) (equal (|poGlobalLinePosn| pos) num)))

defun redundant
redundant(msg,thisPosMsgs) ==
  found := NIL
  if msgNoRep? msg then
    for item in $noRepList repeat
      sameMsg?(msg,item) => return (found := true)
    $noRepList := [msg,$noRepList]
    found or MEMBER(msg,thisPosMsgs)

[msgNoRep? p597]
[sameMsg? p598]
[$noRepList p??]

— defun redundant —
(defun |redundant| (msg thisPosMsgs)
  (prog (found)
    (declare (special |$noRepList|))
    (return
      (progn
        (cond
          ((|msgNoRep?| msg)
            ((lambda (Var9 item)
                (loop
                  (cond
                    ((or (atom Var9) (progn (setq item (car Var9)) nil))
                      (return nil))
                    (t
                      (cond
                        ((|sameMsg?| msg item) (return (setq found t)))))))
            (setq Var9 (cdr Var9))))
          |$noRepList| nil)
            (setq |$noRepList| (list msg |$noRepList|))
            (setq found (member msg thisPosMsgs))))))))

---

defvar $repGuys

— initvars —

(defvar |$repGuys| (list '|noRep| '|rep|))

---

defun msgNoRep?

[getMsgCatAttr p580]

— defun msgNoRep? —

(defun |msgNoRep?| (msg) (eq (|getMsgCatAttr| msg |$repGuys|) '|noRep|))

---
defun sameMsg?

(defun sameMsg? (msg1 msg2)
  (and (equal (getMsgKey msg1) (getMsgKey msg2))
       (equal (getMsgArgL msg1) (getMsgArgL msg2))))

defun processChPosesForOneLine

(defun processChPosesForOneLine (msgList)
  (let ((leaderMsg oldPre posLetter chPosList)
         (declare (special $preLength)))
    (setq chPosList (posPointers msgList))
    (dolist (msg msgList)
      (when (getMsgFTTag? msg) (putFTText msg chPosList))
      (setq posLetter (cdr (assoc (poCharPosn (getMsgPos msg)) chPosList)))
      (setq oldPre (getMsgPrefix msg))
      (setMsgPrefix msg
        (strconc oldPre
          (make-string (- $preLength 4 (size oldPre))) posLetter)))
    (setq leaderMsg (makeLeaderMsg chPosList))
    (nconc msgList (list leaderMsg))))
defun poCharPosn

— defun poCharPosn 0 —

(defun poCharPosn (posn)
  (cdr posn))

defun makeLeaderMsg

makeLeaderMsg chPosList ==
  st := MAKE_-FULL_-CVEC ($preLength- 3)
  oldPos := -1
  for [posNum, posLetter] in reverse chPosList repeat
    st := STRCONC(st, _
      rep(char ".", (posNum - oldPos - 1)), posLetter)
    oldPos := posNum
  [leader,$nopos,'nokey,NIL,NIL,[st] ]

$nopos p251
$preLength p577

— defun makeLeaderMsg —

(defun makeLeaderMsg (chPosList)
  (let (posLetter posNum oldPos st)
    (declare (special $nopos $preLength))
    (setq st (make-string (- $preLength 3))
      oldPos -1)
    ((lambda (Var15 Var14)
      (loop
        (cond
          ((or (atom Var15) (progn (setq Var14 (car Var15)) nil))
            (return nil))
          (t
            (and (consp Var14)
              (progn
                (setq posNum (car Var14))
                (setq posLetter (cdr Var14))
                t)
              (progn
                (setq st
                  (strconc st (_rep ".", (- posNum oldPos 1)), posLetter))
                (setq oldPos posNum)))))
      (setq Var15 (cdr Var15)))))}
(reverse chPosList) nil)
(list 'leader| |$nopos| |nokey| nil nil (list st)))

---

defun posPointers

TPDHERE: getMsgFTTag is nonsense
[poCharPosn p599]
[getMsgPos p581]
[ifcar p??]
[getMsgPos2 p600]
[insertPos p601]
[getMsgFTTag p??]

defun posPointers (msgList)
(let (posLetterList pos ftPosList posList increment pointers)
(declare (special getMsgFTTag))
(setq pointers "ABCDEFGHIJKLMONPQRS")
(setq increment 0)
(dolist (msg msgList)
  (setq pos (poCharPosn (getMsgPos msg)))
  (unless (equal pos (ifcar posList))
    (setq posList (cons pos posList)))
; this should probably read TPDHERE
; (when (eq (getMsgPosTagOb msg) 'fromto))
  (when (eq getMsgFTTag 'fromto)
    (setq ftPosList (cons (poCharPosn (getMsgPos2 msg)) ftPosList))))
(dolist (toPos ftPosList)
  (setq posList (insertPos toPos posList)))
(dolist (pos posList)
  (setq posLetterList
    (cons (cons pos (elt pointers increment)) posLetterList))
  (setq increment (+ increment 1)))
posLetterList))

---

defun getMsgPos2

[getMsgFTTag? p581]
[getMsgPosTagOb p570]
19.2. MESSAGES

[ncBug p590]

—— defun getMsgPos2 ——

(defun getMsgPos2 (msg)
  (if (getMsgFTTag? msg)
      (caddr (getMsgPosTagOb msg))
      (ncBug "not a from to" nil)))

defun insertPos

This function inserts a position in the proper place of a position list. This is used for the 2nd pos of a fromto [done p??]

—— defun insertPos 0 ——

(defun insertPos (newPos posList)
  (let (pos top bot done)
    (setq bot (cons 0 posList))
    (do () (done)
      (setq top (cons (car bot) top))
      (setq bot (cdr bot))
      (setq pos (car bot))
      (setq done
        (cond
          ((< pos newPos) nil)
          ((equal pos newPos) t)
          ((< newPos pos)
            (setq top (cons newPos top))
            t))))
    (cons (cdr (reverse top)) bot)))

———

defun putFTText

[getMsgFTTag? p581]
[poCharPosn p599]
[getMsgPos p581]
[setMsgText p572]
[getMsgText p571]
[getMsgPos2 p600]
defun putFTText

(let (charMarker2 pos2 markingText charMarker pos tag)
  (setq tag (getMsgFTTag? msg))
  (setq pos (poCharPosn (getMsgPos msg)))
  (setq charMarker (cdr (assoc pos chPosList)))
  (cond
    ((eq tag 'from)
     (setq markingText (list " (from " charMarker " and on) "))
     (setq msg (append markingText (getMsgText msg))))
    ((eq tag 'to)
     (setq markingText (list " (up to " charMarker ") "))
     (setq msg (append markingText (getMsgText msg))))
    ((eq tag 'fromto)
     (setq pos2 (poCharPosn (getMsgPos2 msg))
     (setq charMarker2 (cdr (assoc pos2 chPosList)))
     (setq markingText (list " (from " charMarker " up to " charMarker2 " ) "))
     (setq msg (append markingText (getMsgText msg)))))

defun From

This is called from parameter list of nc message functions

(defun From 0)

(defun From|pos| (list 'from pos))

defun To

This is called from parameter list of nc message functions

(defun To 0)

(defun To|pos| (list 'to pos))

defun FromTo

This is called from parameter list of nc message functions

(defun FromTo 0)
(defun FromTo (pos1 pos2) (list 'fromto pos1 pos2))
Chapter 20

The Interpreter Syntax

20.1 syntax assignment

--- assignment.help ---

Immediate, Delayed, and Multiple Assignment

====================================================================
Immediate Assignment
====================================================================

A variable in Axiom refers to a value. A variable has a name beginning with an uppercase or lowercase alphabetic character, "%", or "!". Successive characters (if any) can be any of the above, digits, or "?". Case is distinguished. The following are all examples of valid, distinct variable names:

```
  a  tooBig?  a1B2c3%!?
A    %j    numberOfPoints
beta6  %J  numberOfpoints
```

The " := " operator is the immediate assignment operator. Use it to associate a value with a variable. The syntax for immediate assignment for a single variable is:

```
  variable := expression
```

The value returned by an immediate assignment is the value of expression.

```
a := 1
1
  Type: PositiveInteger
```
The right-hand side of the expression is evaluated, yielding 1. The value is then assigned to a.

```
b := a
1
Type: PositiveInteger
```

The right-hand side of the expression is evaluated, yielding 1. This value is then assigned to b. Thus a and b both have the value 1 after the sequence of assignments.

```
a := 2
2
Type: PositiveInteger
```

What is the value of b if a is assigned the value 2?

```
b
1
Type: PositiveInteger
```

The value of b is left unchanged.

This is what we mean when we say this kind of assignment is immediate. The variable b has no dependency on a after the initial assignment. This is the usual notion of assignment in programming languages such as C, Pascal, and Fortran.

====================================================================
Delayed Assignment
====================================================================

Axiom provides delayed assignment with "==". This implements a delayed evaluation of the right-hand side and dependency checking. The syntax for delayed assignment is

```
variable == expression
```

The value returned by a delayed assignment is the unique value of Void.

```
a == 1
Type: Void
```

```
b == a
Type: Void
```

Using a and b as above, these are the corresponding delayed assignments.

```
a
```
Compiling body of rule a to compute value of type PositiveInteger
1
    Type: PositiveInteger

The right-hand side of each delayed assignment is left unevaluated until
the variables on the left-hand sides are evaluated.

b
Compiling body of rule b to compute value of type PositiveInteger
1
    Type: PositiveInteger

This gives the same results as before. But if we change a to 2

a == 2
Compiled code for a has been cleared.
Compiled code for b has been cleared.
1 old definition(s) deleted for function or rule a
    Type: Void

Then a evaluates to 2, as expected

a
Compiling body of rule a to compute value of type PositiveInteger
2
    Type: PositiveInteger

but the value of b reflects the change to a

b
Compiling body of rule b to compute value of type PositiveInteger
2
    Type: PositiveInteger

====================================================================

Multiple Immediate Assignments
====================================================================

It is possible to set several variables at the same time by using a
tuple of variables and a tuple of expressions. A tuple is a collection
of things separated by commas, often surrounded by parentheses. The
syntax for multiple immediate assignment is

( var1, var2, ..., varN ) := ( expr1, expr2, ..., exprN )

The value returned by an immediate assignment is the value of exprN.

( x, y ) := ( 1, 2 )
2
    Type: PositiveInteger
This sets $x$ to 1 and $y$ to 2. Multiple immediate assignments are parallel in the sense that the expressions on the right are all evaluated before any assignments on the left are made. However, the order of evaluation of these expressions is undefined.

$$(x, y) := (y, x)$$

1
Type: PositiveInteger

$x$

2
Type: PositiveInteger

The variable $x$ now has the previous value of $y$.

$y$

1
Type: PositiveInteger

The variable $y$ now has the previous value of $x$.

There is no syntactic form for multiple delayed assignments.

---

### 20.2 syntax blocks

--- blocks.help ---

A block is a sequence of expressions evaluated in the order that they appear, except as modified by control expressions such as leave, return, iterate, and if-then-else constructions. The value of a block is the value of the expression last evaluated in the block.

To leave a block early, use "=>". For example,

$$i < 0 => x$$

The expression before the "=>' must evaluate to true or false. The expression following the "=>' is the return value of the block.
A block can be constructed in two ways:

1. the expressions can be separated by semicolons and the resulting expression surrounded by parentheses, and
2. the expressions can be written on succeeding lines with each line indented the same number of spaces (which must be greater than zero).

A block entered in this form is called a pile

Only the first form is available if you are entering expressions directly to Axiom. Both forms are available in .input files. The syntax for a simple block of expressions entered interactively is

\[( \text{expression1} ; \text{expression2} ; \ldots ; \text{expressionN} )\]

The value returned by a block is the value of an "=>" expression, or expressionN if no "=>" is encountered.

In .input files, blocks can also be written in piles. The examples given here are assumed to come from .input files.

\[
a := ( \text{i} := \gcd(234,672) ; \text{i} := 2 \times \text{i}^5 - \text{i} + 1 ; \frac{1}{\text{i}} )
\]

\[
1
\]

\[
\frac{1}{23323}
\]

Type: Fraction Integer

In this example, we assign a rational number to a using a block consisting of three expressions. This block is written as a pile. Each expression in the pile has the same indentation, in this case two spaces to the right of the first line.

\[
a := ( i := \gcd(234,672); i := 2 \times i^5 - i + 1; 1 / i )
\]

\[
1
\]

\[
\frac{1}{23323}
\]

Type: Fraction Integer

Here is the same block written on one line. This is how you are required to enter it at the input prompt.

\[
( a := 1; b := 2; c := 3; [a,b,c] )
\]

\[
[1,2,3]
\]

Type: List PositiveInteger

Axiom gives you two ways of writing a block and the preferred way in
an .input file is to use a pile. Roughly speaking, a pile is a block whose constituent expressions are indented the same amount. You begin a pile by starting a new line for the first expression, indenting it to the right of the previous line. You then enter the second expression on a new line, vertically aligning it with the first line. And so on. If you need to enter an inner pile, further indent its lines to the right of the outer pile. Axiom knows where a pile ends. It ends when a subsequent line is indented to the left of the pile or the end of the file.

Also See:
- `)help if`
- `)help repeat`
- `)help while`
- `)help for`
- `)help suchthat`
- `)help parallel`
- `)help lists`

---

## 20.3 system clef

--- clef.help ---

Entering printable keys generally inserts new text into the buffer (unless in overwrite mode, see below). Other special keys can be used to modify the text in the buffer. In the description of the keys below, `^n` means Control-n, or holding the CONTROL key down while pressing "n". Errors will ring the terminal bell.

- `^A/^E` : Move cursor to beginning/end of the line.
- `^F/^B` : Move cursor forward/backward one character.
- `^D` : Delete the character under the cursor.
- `^H, DEL` : Delete the character to the left of the cursor.
- `^K` : Kill from the cursor to the end of line.
- `^L` : Redraw current line.
- `^O` : Toggle overwrite/insert mode. Initially in insert mode. Text added in overwrite mode (including yanks) overwrite existing text, while insert mode does not overwrite.
- `^P/^N` : Move to previous/next item on history list.
- `^R/^S` : Perform incremental reverse/forward search for string on the history list. Typing normal characters adds to the current

---

1 “if” (20.6 p 617) “repeat” (20.10 p 624) “while” (36.1 p 1190) “for” (20.5 p 613) “suchthat” (20.11 p 628) “parallel” (20.9 p 621) “lists” (?) p ??
search string and searches for a match. Typing ^R/^S marks
the start of a new search, and moves on to the next match.
Typing ^H or DEL deletes the last character from the search
string, and searches from the starting location of the last search.
Therefore, repeated DEL’s appear to unwind to the match nearest
the point at which the last ^R or ^S was typed. If DEL is
repeated until the search string is empty the search location
begins from the start of the history list. Typing ESC or
any other editing character accepts the current match and
loads it into the buffer, terminating the search.

^T : Toggle the characters under and to the left of the cursor.
^Y : Yank previously killed text back at current location. Note that
this will overwrite or insert, depending on the current mode.
^U : Show help (this text).
TAB : Perform command completion based on word to the left of the cursor.
Words are deemed to contain only the alphanumeric and the % ! ? _
characters.

NL, CR : returns current buffer to the program.

DOS and ANSI terminal arrow key sequences are recognized, and act like:

up : same as ^P
down : same as ^N
left : same as ^B
right : same as ^F

20.4 syntax collection

--- collection.help ---

====================================================================
Collection -- Creating Lists and Streams with Iterators
====================================================================

All of the loop expressions which do not use the repeat leave or
iterate words can be used to create lists and streams. For example:

This creates a simple list of the integers from 1 to 10:

list := [i for i in 1..10]
[1,2,3,4,5,6,7,8,9,10]
Type: List PositiveInteger

Create a stream of the integers greater than or equal to 1:
stream := [i for i in 1..]
[1,2,3,4,5,6,7,...]
Type: Stream PositiveInteger

This is a list of the prime numbers between 1 and 10, inclusive:

[i for i in 1..10 | prime? i]
[2,3,5,7]
Type: List PositiveInteger

This is a stream of the prime integers greater than or equal to 1:

[i for i in 1.. | prime? i]
[2,3,5,7,11,13,17,...]
Type: Stream PositiveInteger

This is a list of the integers between 1 and 10, inclusive, whose squares are less than 700:

[i for i in 1..10 while i*i < 700]
[1,2,3,4,5,6,7,8,9,10]
Type: List PositiveInteger

This is a stream of the integers greater than or equal to 1 whose squares are less than 700:

[i for i in 1.. while i*i < 700]
[1,2,3,4,5,6,7,...]
Type: Stream PositiveInteger

The general syntax of a collection is

[ collectExpression iterator1 iterator2 ... iteratorN ]

where each iterator is either a for or a while clause. The loop terminates immediately when the end test of any iterator succeeds or when a return expression is evaluated in collectExpression. The value returned by the collection is either a list or a stream of elements, one for each iteration of the collectExpression.

Be careful when you use while to create a stream. By default Axiom tries to compute and display the first ten elements of a stream. If the while condition is not satisfied quickly, Axiom can spend a long (potentially infinite) time trying to compute the elements. Use

)set streams calculate

to change the defaults to something else. This also affects the number of terms computed and displayed for power series. For the purposes of
these examples we have use this system command to display fewer than ten terms.

20.5 syntax for

--- for.help ---

for loops

Axiom provide the for and in keywords in repeat loops, allowing you to integrate across all elements of a list, or to have a variable take on integral values from a lower bound to an upper bound. We shall refer to these modifying clauses of repeat loops as for clauses. These clauses can be present in addition to while clauses (See help while). As with all other types of repeat loops, leave (see help leave) can be used to prematurely terminate evaluation of the loop.

The syntax for a simple loop using for is

for iterator repeat loopbody

The iterator has several forms. Each form has an end test which is evaluated before loopbody is evaluated. A for loop terminates immediately when the end test succeeds (evaluates to true) or when a leave or return expression is evaluated in loopbody. The value returned by the loop is the unique value of Void.

for i in n..m repeat

If for is followed by a variable name, the in keyword and then an integer segment of the form n..m, the end test for this loop is the predicate i > m. The body of the loop is evaluated m-n+1 times if this number is greater than 0. If this number is less than or equal to 0, the loop body is not evaluated at all.

The variable i has the value n, n+1, ..., m for successive iterations of the loop body. The loop variable is a local variable within the loop body. Its value is not available outside the loop body and its value and type within the loop body completely mask any outer definition of a variable with the same name.
for i in 10..12 repeat output(i**3)
1000
1331
1728

Type: Void

The loop prints the values of 10^3, 11^3, and 12^3.

a := [1,2,3]
[1,2,3]

Type: List PositiveInteger

for i in 1..#a repeat output(a.i)
1
2
3

Type: Void

Iterate across this list using "." to access the elements of a list and the # operation to count its elements.

This type of iteration is applicable to anything that uses ".". You can also use it with functions that use indices to extract elements.

m := matrix [ [1,2],[4,3],[9,0] ]
+- -+-
| 1 2 |
| 4 3 |
| 9 0 |
+- -+-

Type: Matrix Integer

Define m to be a matrix.

for i in 1..nrows(m) repeat output row(m.i)
[1,2]
[4,3]
[9,0]

Type: Void

Display the rows of m.

You can iterate with for-loops.

for i in 1..5 repeat
  if odd?(i) then iterate
  output(i)
2
4
Display the even integers in a segment.

====================================================================
for i in n..m by s repeat
====================================================================

By default, the difference between values taken on by a variable in loops such as

    for i in n..m repeat ...

is 1. It is possible to supply another, possibly negative, step value by using the by keyword along with for and in. Like the upper and lower bounds, the step value following the by keyword must be an integer. Note that the loop

    for i in 1..2 by 0 repeat output(i)

will not terminate by itself, as the step value does not change the index from its initial value of 1.

    for i in 1..5 by 2 repeat output(i)
    1
    3
    5

Type: Void

This expression displays the odd integers between two bounds.

    for i in 5..1 by -2 repeat output(i)
    5
    3
    1

Type: Void

Use this to display the numbers in reverse order.

====================================================================
for i in n.. repeat
====================================================================

If the value after the ".." is omitted, the loop has no end test. A potentially infinite loop is thus created. The variable is given the successive values \( n, n+1, n+2, \ldots \) and the loop is terminated only if a leave or return expression is evaluated in the loop body. However, you may also add some other modifying clause on the repeat, for example, a while clause, to stop the loop.
for i in 15.. while not prime?(i) repeat output(i)
15
16

Type: Void

This loop displays the integers greater than or equal to 15 and less than the first prime number greater than 15.

for x in l repeat

Another variant of the for loop has the form:

for x in list repeat loopbody

This form is used when you want to iterate directly over the elements of a list. In this form of the for loop, the variable x takes on the value of each successive element in l. The end test is most simply stated in English: "are there no more x in l?"

l := [0, -5, 3]

Type: List Integer

for x in l repeat output(x)
0
-5
3

Type: Void

This displays all of the elements of the list l, one per line.

Since the list constructing expression

expand [n..m]

creates the list

[n, n+1, ..., m]

you might be tempted to think that the loops

for i in n..m repeat output(i)

and

for x in expand [n..m] repeat output(x)

are equivalent. The second form first creates the expanded list
(no matter how large it might be) and then does the iteration. The first form potentially runs in much less space, as the index variable i is simply incremented once per loop and the list is not actually created. Using the first form is much more efficient.

Of course, sometimes you really want to iterate across a specific list. This displays each of the factors of 2400000:

```axiom
def f in factors(factor(2400000)) repeat output(f)
[factor= 2, exponent= 8]
[factor= 3, exponent= 1]
[factor= 5, exponent= 5]
Type: Void
```

---

## 20.6 Syntax If

---

### if.help ---

If-then-else

Like many other programming languages, Axiom uses the three keywords if, then, and else to form conditional expressions. The else part of the conditional is optional. The expression between the if and then keywords is a predicate: an expression that evaluates to or is convertible to either true or false, that is, a Boolean.

The syntax for conditional expressions is

```
if predicate then expression1 else expression2
```

where the "else expression2" part is optional. The value returned from a conditional expression is expression1 if the predicate evaluates to true and expression2 otherwise. If no else clause is given, the value is always the unique value of Void.

An if-then-else expression always returns a value. If the else clause is missing then the entire expression returns the unique value of Void. If both clauses are present, the type of the value returned by if is obtained by resolving the types of the values of the two clauses.

The predicate must evaluate to, or be convertible to, an object of type Boolean: true or false. By default, the equal sign "=" creates an equation.
\begin{equation}
  x + 1 = y
\end{equation}

\textbf{Type: Equation Polynomial Integer}

This is an equation, not a boolean condition. In particular, it is an object of type Equation Polynomial Integer.

However, for predicates in if expressions, Axiom places a default target type of Boolean on the predicate and equality testing is performed. Thus you need not qualify the "=" in any way. In other contexts you may need to tell Axiom that you want to test for equality rather than create an equation. In these cases, use "@" and a target type of Boolean.

The compound symbol meaning "not equal" in Axiom is "\texttt{~=}". This can be used directly without a package call or a target specification. The expression "\texttt{a ~=} b" is directly translated to "\texttt{not(a = b)}".

Many other functions have return values of type Boolean. These include <, <=, >, >=, \texttt{~=}, and \texttt{member?}. By convention, operations with names ending in "\texttt{?}" return Boolean values.

The usual rules for piles are suspended for conditional expressions. In .input files, the then and else keywords can begin in the same column as the corresponding if by may also appear to the right. Each of the following styles of writing if-then-else expressions is acceptable:

\begin{verbatim}
if i>0 then output("positive") else output("nonpositive")

if i>0 then output("positive")
   else output("nonpositive")

if i>0 then output("positive")
   else output("nonpositive")

if i>0
   then output("positive")
   else output("nonpositive")

if i>0
   then output("positive")
   else output("nonpositive")
\end{verbatim}

A block can follow the then or else keywords. In the following two assignments to \texttt{a}, the then and else clauses each are followed by two line piles. The value returned in each is the value of the second line.

\begin{verbatim}
a :=
   if i > 0 then
      j := \texttt{sin(i * pi())}
\end{verbatim}
exp(j + 1/j)
else
  j := cos(i * 0.5 * pi())
  log(abs(j)**5 + i)

a :=
  if i > 0
    then
      j := sin(i * pi())
      exp(j + 1/j)
    else
      j := cos(i * 0.5 * pi())
      log(abs(j)**5 + i)

These are both equivalent to the following:

a :=
  if i > 0 then (j := sin(i * pi()); exp(j + 1/j))
  else (j := cos(i * 0.5 * pi()); log(abs(j)**5 + i))

— iterate.help —

iterate in loops

Axiom provides an iterate expression that skips over the remainder of a loop body and starts the next loop execution. We first initialize a counter.

i := 0
0

Type: NonNegativeInteger

Display the even integers from 2 to 5:

repeat
  i := i + 1
  if i > 5 then leave
  if odd?(i) then iterate
  output(i)
20.8 syntax leave

--- leave.help ---

The leave keyword is often more useful in terminating a loop. A leave causes control to transfer to the expression immediately following the loop. As loops always return the unique value of Void, you cannot return a value with leave. That is, leave takes no argument.

```plaintext
f() ==
  i := 1
  repeat
    if factorial(i) > 1000 then leave
    i := i + 1
  i

Type: Void
```

This example is a modification of the last example in the previous section. Instead of using return we'll use leave.

```plaintext
f()

7

Type: PositiveInteger
```

The loop terminates when factorial(i) gets big enough. The last line of the function evaluates to the corresponding "good" value of i and the function terminates, returning that value.

You can only use leave to terminate the evaluation of one loop. Let's consider a loop within a loop, that is, a loop with a nested loop. First, we initialize two counter variables.

```plaintext
(i,j) := (1,1)

1

Type: PositiveInteger
```
Nested loops must have multiple leave expressions at the appropriate nesting level. How would you rewrite this so \((i + j) > 10\) is only evaluated once?

---

**leave vs => in loop bodies**

---

Compare the following two loops:

```plaintext
i := 1
repeat
  i := i + 1
  i > 3 => i
  output(i)
```

In the example on the left, the values 2 and 3 for \(i\) are displayed but then the "\(=>\)" does not allow control to reach the call to `output` again. The loop will not terminate until you run out of space or interrupt the execution. The variable \(i\) will continue to be incremented because the "\(=>\)" only means to leave the block, not the loop.

In the example on the right, upon reaching 4, the leave will be executed, and both the block and the loop will terminate. This is one of the reasons why both "\(=>\)" and `leave` are provided. Using a while clause with the "\(=>\)" lets you simulate the action of `leave`.

---

**20.9 syntax parallel**

---

**parallel.help**

---

**parallel iteration**

Sometimes you want to iterate across two lists in parallel, or perhaps
you want to traverse a list while incrementing a variable.

The general syntax of a repeat loop is

\[
\text{iterator1, iterator2, \ldots, iteratorN repeat loopbody}
\]

where each iterator is either a for or a while clause. The loop
terminates immediately when the end test of any iterator succeeds or
when a leave or return expression is evaluated in loopbody. The value
returned by the loop is the unique value of Void.

```
1 := [1,3,5,7]
[1,3,5,7]
   Type: List PositiveInteger

m := [100,200]
[100,200]
   Type: List PositiveInteger

sum := 0
0
   Type: NonNegativeInteger
```

Here we write a loop to iterate across two lists, computing the sum
of the pairwise product of the elements:

```
for x in l for y in m repeat
  sum := sum + x\*y
  Type: Void
```

The last two elements of l are not used in the calculation because
m has two fewer elements than l.

```
sum
700
   Type: NonNegativeInteger
```

This is the "dot product".

Next we write a loop to compute the sum of the products of the loop
elements with their positions in the loop.

```
l := [2,3,5,7,11,13,17,19,23,29,31,37]
[2,3,5,7,11,13,17,19,23,29,31,37]
   Type: List PositiveInteger

sum := 0
0
   Type: NonNegativeInteger
```
for i in 0.. for x in l repeat sum := i * x
Type: Void

Here looping stops when the list l is exhausted, even though the for i in 0.. specifies no terminating condition.

sum
407
Type: NonNegativeInteger

When "|" is used to qualify any of the for clauses in a parallel iteration, the variables in the predicates can be from an outer scope or from a for clause in or to the left of the modified clause.

This is correct:

for i in 1..10 repeat
  for j in 200..300 | ood? (i+j) repeat
    output [i,j]

But this is not correct. The variable j has not been defined outside the inner loop:

for i in 1..01 | odd? (i+j) repeat -- wrong, j not defined
  for j in 200..300 repeat
    output [i,j]

It is possible to mix several of repeat modifying clauses on a loop:

for i in 1..10
  for j in 151..160 | odd? j
    while i + j < 160 repeat
      output [i,j]
[1,151]
[3,153]
Type: Void

Here are useful rules for composing loop expressions:

1. while predicates can only refer to variables that are global (or in an outer scope) or that are defined in for clauses to the left of the predicate.
2. A "such that" predicate (somthing following "|") must directly follow a for clause and can only refer to variables that are global (or in an outer scope) or defined in the modified for clause or any for clause to the left.
20.10 syntax repeat

— repeat.help —

A loop is an expression that contains another expression, called the loop body, which is to be evaluated zero or more times. All loops contain the repeat keyword and return the unique value of Void. Loops can contain inner loops to any depth.

The most basic loop is of the form

    repeat loopbody

Unless loopbody contains a leave or return expression, the loop repeats forever. The value returned by the loop is the unique value of Void.

Axiom tries to determine completely the type of every object in a loop and then to translate the loop body to Lisp or even to machine code. This translation is called compilation.

If Axiom decides that it cannot compile the loop, it issues a message stating the problem and then the following message:

    We will attempt to step through and interpret the code

It is still possible that Axiom can evaluate the loop but in interpret-code mode.

A return expression is used to exit a function with a particular value. In particular, if a return is in a loop within the function, the loop is terminated whenever the return is evaluated.

```plaintext
f() ==
  i := 1
  repeat
    if factorial(i) > 1000 then return i
  i := i + 1
  Type: Void

f()
```
Type: Void

When factorial(i) is big enough, control passes from inside the loop all the way outside the function, returning the value of i (so we think). What went wrong? Isn’t it obvious that this function should return an integer? Well, Axiom makes no attempt to analyze the structure of a loop to determine if it always returns a value because, in general, this is impossible. So Axiom has this simple rule: the type of the function is determined by the type of its body, in this case a block. The normal value of a block is the value of its last expression, in this case, a loop. And the value of every loop is the unique value of Void. So the return type of f is Void.

There are two ways to fix this. The best way is for you to tell Axiom what the return type of f is. You do this by giving f a declaration

\[ f(): \rightarrow \text{Integer} \]

prior to calling for its value. This tells Axiom "trust me -- an integer is returned". Another way is to add a dummy expression as follows.

\[ f() == \]
\[ i := 1 \]
\[ \text{repeat} \]
\[ \text{if factorial(i) > 1000 then return i} \]
\[ i := i + 1 \]
\[ 0 \]

Type: Void

Note that the dummy expression will never be evaluated but it is the last expression in the function and will determine the return type.

\[ f() \]
\[ 7 \]

Type: PositiveInteger

====================================================================
leave in loops
====================================================================

The leave keyword is often more useful in terminating a loop. A leave causes control to transfer to the expression immediately following the loop. As loops always return the unique value of Void, you cannot return a value with leave. That is, leave takes no argument.

\[ f() == \]
\[ i := 1 \]
\[ \text{repeat} \]
\[ \text{if factorial(i) > 1000 then leave} \]
\[ i := i + 1 \]
This example is a modification of the last example in the previous section. Instead of using return we'll use leave.

\[ f() \]

Type: PositiveInteger

The loop terminates when factorial(i) gets big enough. The last line of the function evaluates to the corresponding "good" value of i and the function terminates, returning that value.

You can only use leave to terminate the evaluation of one loop. Let's consider a loop within a loop, that is, a loop with a nested loop. First, we initialize two counter variables.

\[(i,j) := (1,1)\]

Type: PositiveInteger

\[
\text{repeat}
\begin{align*}
\text{repeat} & \\
& \text{if } (i + j) > 10 \text{ then leave} \\
& \quad j := j + 1 \\
& \text{if } (i + j) > 10 \text{ then leave} \\
& \quad i := i + 1 \\
\end{align*}
\]

Type: Void

Nested loops must have multiple leave expressions at the appropriate nesting level. How would you rewrite this so \((i + j) > 10\) is only evaluated once?

---

leave vs \(=>\) in loop bodies
---

Compare the following two loops:

\[
\begin{align*}
& \text{i := 1} \\
& \text{repeat} \\
& \quad \text{i := i + 1} \\
& \quad \text{i > 3 => i} \\
& \text{output(i)}
\end{align*}
\]

\[
\begin{align*}
& \text{i := 1} \\
& \text{repeat} \\
& \quad \text{i := i + 1} \\
& \text{i > 3 => i} \\
& \text{output(i)}
\end{align*}
\]

In the example on the left, the values 2 and 3 for i are displayed but then the "\(=>\)" does not allow control to reach the call to output again. The loop will not terminate until you run out of space or interrupt the execution. The variable i will continue to be incremented because the
"=>" only means to leave the block, not the loop.

In the example on the right, upon reaching 4, the leave will be executed, and both the block and the loop will terminate. This is one of the reasons why both "=" and leave are provided. Using a while clause with the "=" lets you simulate the action of leave.

iterate in loops

Axiom provides an iterate expression that skips over the remainder of a loop body and starts the next loop execution. We first initialize a counter.

\begin{verbatim}
i := 0
\end{verbatim}

Type: NonNegativeInteger

Display the even integers from 2 to 5:

\begin{verbatim}
repeat
  i := i + 1
  if i > 5 then leave
  if odd?(i) then iterate
  output(i)
2
4
\end{verbatim}

Type: Void

Also See:
- )help blocks
- )help if
- )help while
- )help for
- )help suchthat
- )help parallel
- )help lists

\footnote{blocks" (20.2 p 608) "if" (20.6 p 617) "while" (36.1 p 1190) "for" (20.5 p 613) "suchthat" (20.11 p 628) "parallel" (20.9 p 621) "lists" (?? p ??)
20.11 syntax suchthat

--- suchthat.help ---

A for loop can be followed by a "|" and then a predicate. The predicate qualifies the use of the values from the iterator that follows the for. Think of the vertical bar "|" as the phrase "such that".

for n in 0..4 | odd? n repeat output n
1
3

Type: Void

This loop expression prints out the integers n in the given segment such that n is odd.

A for loop can also be written

for iterator | predicate repeat loopbody

which is equivalent to:

for iterator repeat if predicate then loopbody else iterate

The predicate need not refer only to the variable in the for clause. Any variable in an outer scope can be part of the predicate.

for i in 1..50 repeat
  for j in 1..50 | factorial(i+j) < 25 repeat
    output [i,j]
[1,1]
[1,2]
[1,3]
[2,1]
[2,2]
[3,1]

Type: Void

---
20.12 syntax syntax

— syntax.help —

The Axiom Interactive Language has the following features documented here.

More information is available by typing

)`help feature`

where feature is one of:

- assignment -- Immediate and delayed assignments
- blocks -- Blocks of expressions
- collection -- creating lists with iterators
- for -- for loops
- if -- If-then-else statements
- iterate -- using iterate in loops
- leave -- using leave in loops
- parallel -- parallel iterations
- repeat -- repeat loops
- suchthat -- suchthat predicates
- while -- while loops

20.13 syntax while

— while.help —

====================================================================
while loops
====================================================================

The repeat in a loop can be modified by adding one or more while clauses. Each clause contains a predicate immediately following the while keyword. The predicate is tested before the evaluation of the body of the loop. The loop body is evaluated whenever the predicate in a while clause is true.

The syntax for a simple loop using while is

while predicate repeat loopbody
The predicate is evaluated before loopbody is evaluated. A while loop terminates immediately when predicate evaluates to false or when a leave or return expression is evaluated. See \texttt{parse repeat} for more information on leave and return.

Here is a simple example of using while in a loop. We first initialize the counter.

\begin{verbatim}
i := 1
1
Type: PositiveInteger

while i < 1 repeat
    output "hello"
i := i + 1
Type: Void
\end{verbatim}

The steps involved in computing this example are

1. set \( i \) to 1
2. test the condition \( i < 1 \) and determine that it is not true
3. do not evaluate the loop body and therefore do not display "hello"

\begin{verbatim}
(x, y) := (1, 1)
1
Type: PositiveInteger
\end{verbatim}

If you have multiple predicates to be tested use the logical and operation to separate them. Axiom evaluates these predicates from left to right.

\begin{verbatim}
while x < 4 and y < 10 repeat
    output [x, y]
x := x + 1
y := y + 2
[1,1]
[2,3]
[3,5]
Type: Void
\end{verbatim}

A leave expression can be included in a loop body to terminate a loop even if the predicate in any while clauses are not false.

\begin{verbatim}
(x, y) := (1, 1)
1
Type: PositiveInteger
\end{verbatim}

\begin{verbatim}
while x < 4 and y < 10 repeat
    if x + y > 7 then leave
    output [x, y]
\end{verbatim}
20.13. SYNTAX WHILE

\[
\begin{align*}
x & := x + 1 \\
y & := y + 2 \\
[1,1] & \\
[2,3] & \\
\text{Type: Void}
\end{align*}
\]

\[\]

---
Chapter 21

Abstract Syntax Trees (ptrees)

Abstract Syntax Trees

These functions create and examine abstract syntax trees. These are called pform, for short.

!! This file also contains constructors for concrete syntax, although they should be somewhere else.

THE PFORM DATA STRUCTURE
Leaves: [hd, tok, pos]
Trees: [hd, tree, tree, ...]
hd is either an id or (id . alist)

defun Construct a leaf token

The tokConstruct function is a constructor and selectors for leaf tokens. A leaf token looks like [head, token, position] where head is either an id or (id . alist)

[ifcar p??]
[pfNoPosition? p634]
[ncPutQ p638]

— defun tokConstruct —

(defun |tokConstruct| (head token &rest position)
  (let (result)
    (setq result (cons head token))
    (cond
      ((ifcar position)
        (cond
          ((pfNoPosition? (car position)) result)
          (cond
            ((|pfNoPosition?| (car position)) result))));

633
(t (incPutQ result 'posn (car position)) result)))
(t result)))

defun Return a part of a node
[ifcar p??]

— defun pfAbSynOp —

(defun pfAbSynOp (form)
(let (hd)
  (setq hd (car form))
  (or (ifcar hd) hd)))

defun Compare a part of a node
[eqcar p??]

— defun pfAbSynOp? —

(defun pfAbSynOp? (form op)
(let (hd)
  (setq hd (car form))
  (or (eq hd op) (eqcar hd op))))

defun pfNoPosition?
[poNoPosition? p635]

— defun pfNoPosition? —

(defun pfNoPosition? (pos)
  (poNoPosition? pos))
defun poNoPosition?
[eqcar p?]
  — defun poNoPosition? 0 —
(defun poNoPosition? (pos)
  (eqcar pos 'noposition!))

defun tokType
[ncTag p637]
  — defun tokType —
(defun tokType (x) (ncTag x))

defun tokPart
  — defun tokPart 0 —
(defun tokPart (x) (cdr x))

defun tokPosn
[qassq p??]
[ncAlist p637]
[pfNoPosition p636]
  — defun tokPosn —
(defun tokPosn (x)
  (let (a)
    (setq a (qassq 'posn (ncAlist x)))
    (cond
(a (cdr a))
(t (\|pfNoPosition\|)))

defun pfNoPosition
[poNoPosition p636]
    — defun pfNoPosition —
    (defun |pfNoPosition| () (|poNoPosition|))

defun poNoPosition
[$nopos p251]
    — defun poNoPosition 0 —
    (defun |poNoPosition| ()
        (declare (special |$nopos|))
        |$nopos|)
Chapter 22

Attributed Structures

For objects which are pairs where the CAR field is either just a tag (an identifier) or a pair which is the tag and an association list.

`defun ncTag`

Pick off the tag [ncBug p590] [qcar p??] [identp p1197]

```lisp
(defun ncTag (x)
  (cond
    ((null (consp x)) (ncBug "bad object" nil))
    (t
      (setq x (qcar x))
      (cond
        ((identp x) x)
        ((null (consp x)) (ncBug "bad object" nil))
        (t (qcar x))))))
```

---

defun ncAlist

Pick off the property list [ncBug p590] [qcar p??] [identp p1197] [qcdr p??]

637
— defun ncAlist —

(defun ncAlist (x)
  (cond
   ((null (consp x)) (ncBug "bad object" nil))
   (t
    (setq x (qcar x))
    (cond
     ((identp x) nil)
     ((null (consp x)) (ncBug "bad object" nil))
     (t (qcdr x))))))

defun ncEltQ

Get the entry for key k on x’s association list

[qassq p??]
[ncAlist p637]
[ncBug p590]

— defun ncEltQ —

(defun ncEltQ (x k)
  (let (r)
    (setq r (qassq k (ncAlist x)))
    (cond
     ((null r) (ncBug "Association list search failed on %1" (list k)))
     (t (cdr r)))))

defun ncPutQ

;-- Put (k . v) on the association list of x and return v
;-- case1: ncPutQ(x,k,v) where k is a key (an identifier), v a value
;-- put the pair (k . v) on the association list of x and return v
;-- case2: ncPutQ(x,k,v) where k is a list of keys, v a list of values
;-- equivalent to [ncPutQ(x,key,val) for key in k for val in v]
;ncPutQ(x,k,v) ==
  ; LISTP k =>
  ; for key in k for val in v repeat ncPutQ(x,key,val)
  ; v
  ; r := QASSQ(k,ncAlist x)
(defun |ncPutQ| (x k v)
  (let (r)
    (cond
      ((listp k)
       ((lambda (Var1 key Var2 val)
          (loop
            (cond
              ((or (atom Var1)
                 (progn (setq key (car Var1)) nil)
                 (atom Var2)
                 (progn (setq val (car Var2)) nil))
              (return nil))
              (t
               (|ncPutQ| x key val)))
            (setq Var1 (cdr Var1))
            (setq Var2 (cdr Var2)))))
      k nil v nil)
    v)
    (t
     (setq r (qassq k (|ncAlist| x)))
     (cond
      ((null r)
       (setq r (cons (cons k v) (|ncAlist| x)))
       (rplaca x (cons (|ncTag| x) r)))
      (t
       (rplacd r v))))
    v)))
Special Category Names

defvar $EmptyMode

The CONTAINED predicate is used to walk internal structures such as modmaps to see
if the $X$ object occurs within $Y$. One particular use is in a function called isPartialMode
to decide if a modemap is only partially complete. If this is true then the modemap will
contain the constant $\text{EmptyMode}$. So the call ends up being CONTAINED $\text{EmptyMode}$ $Y$.

— initvars —

(defvar $EmptyMode| '|$EmptyMode|)

——

defvar $AnonymousFunction

— initvars —

(defvar $AnonymousFunction| '(|AnonymousFunction|))

——

defvar $Any

— initvars —

(defvar $Any| '(|Any|))

——

defvar $BFtag

— initvars —

(defvar $BFtag| '|:BF:|)
defvar $Boolean

— initvars —

(defvar $Boolean '(|Boolean|))

—

defvar $Category

— initvars —

(defvar $Category '(|Category|))

—

defvar $Domain

— initvars —

(defvar $Domain '(|Domain|))

—

defvar $Exit

— initvars —

(defvar $Exit '(|Exit|))

—

defvar $Expression

— initvars —
defvar $Expression  `(|OutputForm|))

---

defvar $OutputForm

---

defvar $BigFloat

---

defvar $Float

---

defvar $DoubleFloat
defvar $FontTable

   — initvars —

(defvar |$FontTable| '(|FontTable|))

defvar $Integer

   — initvars —

(defvar |$Integer| '(|Integer|))

defvar $ComplexInteger

   — initvars —

(defvar |$ComplexInteger| (LIST '|Complex| |$Integer|))

defvar $Mode

   — initvars —

(defvar |$Mode| '(|Mode|))

defvar $NegativeInteger

   — initvars —
(defvar $NegativeInteger '([NegativeInteger]))

-----

defvar $NonNegativeInteger

— initvars —

(defvar $NonNegativeInteger '([NonNegativeInteger]))

-----

defvar $NonPositiveInteger

— initvars —

(defvar $NonPositiveInteger '([NonPositiveInteger]))

-----

defvar $PositiveInteger

— initvars —

(defvar $PositiveInteger '([PositiveInteger]))

-----

defvar $RationalNumber

— initvars —

(defvar $RationalNumber '([Fraction ([Integer])))
defvar $String

    — initvars —

(defvar $String '([String]))

-----

defvar $StringCategory

    — initvars —

(defvar $StringCategory '([StringCategory]))

-----

defvar $Symbol

    — initvars —

(defvar $Symbol '([Symbol]))

-----

defvar $Void

    — initvars —

(defvar $Void '([Void]))

-----

defvar $QuotientField

    — initvars —
(defvar $QuotientField 'Fraction)

---

defvar $FunctionalExpression

---

(defvar $defaultFunctionTargets '())

;; Old names

defvar $SmallInteger

---

(defvar $SmallInteger '(SingleInteger))

;; New Names

defvar $SingleFloat

---

(defvar $SingleFloat '(SingleFloat))
defvar $DoubleFloat

— initvars —

(defvar |$DoubleFloat| '([DoubleFloat]))

defvar $SingleInteger

— initvars —

(defvar |$SingleInteger| '([SingleInteger]))
Chapter 23

Function Selection

New Selection of Modemaps

selection of applicable modemaps is done in two steps:
  first it tries to find a modemap inside an argument domain, and if
this fails, by evaluation of pattern modemaps
the result is a list of functions with signatures, which have the
following form:
  [sig,elt,cond] where
    sig is the signature gained by evaluating the modemap condition
    elt is the slot number to get the implementation
    cond are runtime checks which are the results of evaluating the
    modemap condition

the following flags are used:
  $Coerce is NIL, if function selection is done which requires exact
matches (e.g. for coercion functions)
  if $SubDom is true, then runtime checks have to be compiled

defun ofCategory

[identp p1197]
[ofCategory p649]
[hasCaty p650]
[$Subst p??]
[$hope p??]

— defun ofCategory —

(defun ofCategory (dom cat)
  (let (($Subst |$Subst| |$hope|)
        (declare (special |$Subst| |$hope|)))

649
(cond
  ((identp dom) nil)
  ((and (listp cat) (eq (car cat) '|Join|))
    (every #'(lambda (c) (|ofCategory| dom c)) (cdr cat)))
  (t (not (eq (|hasCaty| dom cat nil) '|failed|))))

---

defun isPartialMode

defun isPartialMode

The isPartialMode function tests whether m contains $EmptyMode. The constant $EmptyMode evaluates to $EmptyMode. This constant is inserted in a modemap during compile time if the modemap is not yet complete.

— defun isPartialMode —

defun isPartialMode (m)
  (declare (special $EmptyMode))
  (contained $EmptyMode m))

---

defun hasCaty

This calls hasCat, which looks up a hashtable and returns:

1. T, NIL or a (has x1 x2) condition, if cat is not parameterized
2. a list of pairs (argument to cat, condition) otherwise

then the substitution sl is augmented, or the result is 'failed [hasAttSig p656]

[subCopy p653]
[constructSubst p664]
[hasSig p653]
[hasAtt p654]
[hasCat p654]
[opOf p654]
[mkDomPvar p663]
[domArg p652]
[augmentSub p652]
[domArg2 p652]
[unifyStruct p658]
(defun hasCaty (d cat sl)
  (let (x y S z cond sp dom zp s1 ncond i)
    (declare (special $domPvar))
    (cond
      ((and (consp cat) (eq (qcar cat) 'category) (consp (qcdr cat)))
       (hasAttSig d (subCopy (qcddr cat) (constructSubst d)) sl))
      ((and (consp cat) (eq (qcar cat) 'signature) (consp (qcdr cat)))
       (hasSig d (qcadr cat) (subCopy (qcaddr cat) (constructSubst d)) sl))
      ((and (consp cat) (eq (qcar cat) 'attribute)
          (consp (qcdr cat)))
       (hasAtt d (subCopy (qcadr cat) (constructSubst d)) nil))
      ((setq x (hasCat (opOf d) (opOf cat)))
       (cond
        ((setq y (ifcdr cat))
         (setq s (constructSubst d))
         (do ((next x (cdr next)) (endtest nil (null (eq s1 'failed))))
             ((or (atom next) endtest) nil)
           (setq z (caar next))
           (setq cond (cadr next))
           (setq sp
             (loop for item in s
                   collect (cons (car item) (mkDomPvar (car item) (cdr item) z y))))
           (when $domPvar)
           (setq i -1)
           (setq dom
             (cons (car d)
               (loop for arg in (rest d)
                     collect (domArg arg (incf i) z y))))
           (setq s1 (augmentSub $domPvar dom (copy sl)))
           (setq zp
             (loop for a in z
collect (domArg2 a s sp)))
           (setq s1 (unifyStruct y zp (copy sl)))
           (cond
            ((null (eq s1 'failed))
             (setq s1
               (cond
                ((atom cond) s1)
                (t
                 (setq ncond (subCopy cond s))
                 (cond
                  ((and (consp ncond) (eq (qcar ncond) 'has))
                   (consp (qcdr ncond))
                   (equal (qcadr ncond) d)
                   (consp (qcddr ncond))
                   (eq (qcdddr ncond) nil))
                  (setq zp
                    (loop for a in z
collect (domArg2 a s sp)))
                  (setq s1 (unifyStruct y zp (copy sl)))
                  (cond
                   ((null (eq s1 'failed))
                    (setq s1
                      (cond
                       ((atom cond) s1)
                       (t
                        (setq ncond (subCopy cond s))
                        (cond
                         ((and (consp ncond) (eq (qcar ncond) 'has))
                          (consp (qcdr ncond))
                          (equal (qcadr ncond) d)
                          (consp (qcddr ncond))
                          (eq (qcdddr ncond) nil))))
```
(equal (qcaddr ncond) cat))
  '|failed|
  (t (|hasCaty1| ncond sl)))))))
  (t nil))))
  s1)
((atom x) sl)
(t
  (setq ncond (|subCopy| x (|constructSubst| d)))
  (cond
    (((and (consp ncond) (eq (qcar ncond) '|has|) (consp (qcdr ncond))
      (equal (qcadr ncond) d) (consp (qcddr ncond))
      (eq (qcdddr ncond) nil) (equal (qcaddr ncond) cat))
      '|failed|
    (t (|hasCaty1| ncond sl)))))))
  (t '|failed|)))

defun domArg
[$FormalMapVariableList p??]
  — defun domArg —
(defun |domArg| (type i subs y)
  (let (p)
    (declare (special |$FormalMapVariableList|))
    (if (setq p (member (elt |$FormalMapVariableList| i) subs))
      (elt y (- (|#| subs) (|#| p)))
      type)))

defun domArg2
[isSharpVar p1064]
[subCopy p??]
[|domPvar| p274]
  — defun domArg2 —
(defun |domArg2| (arg sl1 sl2)
  (declare (special |$domPvar|))
  (cond
    (((isSharpVar| arg) (|subCopy| arg sl1))}
defun hasSig

The function hasSig tests whether domain dom has function foo with signature sig under
substitution sl. 

(defun hasSig (dom foo sig sl)
  (let ((|$domPvar| fun s0 p x cond s))
    (declare (special |$domPvar|))
    (cond
      ((setq fun (|constructor?| (car dom)))
        (setq s0 (|constructSubst| dom))
        (cond
          ((setq p (assq foo (|getOperationAlistFromLisplib| (car dom))))
            (do ((next (cdr p) (cdr next))
                 (endtest nil (null (eq s '|failed|))))
                (or (atom next) endtest) nil)
            (setq x (caar next))
            (setq cond (caddar next))
            (setq s
              (cond
                ((atom cond) (copy sl))
                ((and (consp cond) (eq (qcar cond) '|has|))
                  (consp (qcdr cond)) (consp (qcddr cond)))
                (eq (qcdr (qcddr cond)) nil))
                (|hasCate| (|subCopy| (qcadr cond) s0)
                  (|subCopy| (qcaddr cond) s0)
                  (copy sl)))
              (and (consp cond)
                (or (eq (qcar cond) '|and|) (eq (qcar cond) '|and|))
                (|hasSigAnd| (qcdr cond) s0 sl))))
            ))
      )
    ))
  )
defun hasAtt

The hasAtt function tests whether dom has attribute att under sl needs s0 similar to hasSig.

(defun hasAtt (dom att sl)
  (let ((domPvar fun atts u x cond s)
        (declare (special domPvar)))
    (cond
      ((setq fun (car dom))
       (cond
         ((setq atts (subCopy (getdatabase fun 'attributes) (constructSubst dom)))
          (do ((next atts (cdr next) (endtest nil (null (eq s 'failed))))
               (or (atom next) endtest) nil)
              (setq x (caar next))
              (setq cond (cdar next))
              (setq s (unifyStruct x att (copy sl)))
              (cond
               ((and (null (atom cond)) (null (eq s 'failed)))
                (setq s (hasCatExpression cond s))))))))

defun hasAtt —

(defun hasAtt (dom att sl)
  (let ((domPvar fun atts u x cond s)
        (declare (special domPvar)))
    (cond
      ((setq fun (car dom))
       (cond
         ((setq atts (subCopy (getdatabase fun 'attributes) (constructSubst dom)))
          (cond
           ((consp (setq u (getInfovec (car dom)))
             (do ((next atts (cdr next))
                  (endtest nil (null (eq s 'failed))))
                (setq x (caar next))
                (setq cond (cdar next))
                (setq s (unifyStruct x att (copy sl)))
                (cond
                 ((and (null (atom cond)) (null (eq s 'failed)))
                  (setq s (hasCatExpression cond s))))))))
s)
(t
  (do ((next atts (cdr next))
        (endtest nil (null (eq s 'failed))))
      ((or (atom next) endtest) nil)
    (setq x (caar next))
    (setq cond (cadar next))
    (setq s (unifyStruct x att (copy sl)))
    (cond
      ((and (null (atom cond)) (null (eq s 'failed)))
       (setq s (hasCatExpression cond s))))
    s))
  (t 'failed))
  (t 'failed)))

---

defun hasSigAnd

[hasCate p663]
[subCopy p??]
[keyedSystemError p??]

  defun hasSigAnd —

  (defun hasSigAnd| (andCls s0 sl)
    (let (sa dead)
      (setq sa 'failed')
      (loop for cls in andCls
        do
          (when dead (return))
          (setq sa
            (cond
              ((atom cls) (copy sl))
              ((and (consp cls) (eq (qcar cls) 'has') (consp (qcdr cls))
                 (consp (qcdrr cls)) (eq (qcdrrr cls) nil))
               (hasCate (subCopy (qcadr cls) s0)
                         (subCopy (qcaddr cls) s0)
                         (copy sl)))
              (t
               (keyedSystemError
                 "Unexpected error or improper call to system function %1: %2"
                 (list "hasSigAnd" "unexpected condition for signature")))
          (when (eq sa 'failed) (setq dead t)))
      sa))

---
defun hasSigOr

(hasCate p663)
(hasSigAnd p655)
(keyedSystemError p??)

— defun hasSigOr —

(defun |hasSigOr| (orCls s0 sl)
(let (sa found)
  (setq sa '|failed|)
  (loop for cls in orCls
        until found
        do
          (setq sa
            (cond
              ((atom cls) (copy sl))
              ((and (consp cls) (eq (qcar cls) '|has|) (consp (qcdr cls))
                  (consp (qcdrr cls)) (eq (qcdddr cls) nil))
                (hasCate| (|subCopy| (qcadr cls) s0)
                  (|subCopy| (qcaddr cls) s0)
                  (copy sl)))
              ((and (consp cls)
                (or (eq (qcar cls) 'and) (eq (qcar cls) '|and|)))
                (hasSigAnd| (qcdr cls) s0 sl))
              (t
                (keyedSystemError
                  "Unexpected error or improper call to system function %1: %2"
                  (list "hasSigOr" "unexpected condition for signature")))
            ))
          (unless (eq sa '|failed|) (setq found t)))
  sa))

—

defun hasAttSig

The argument d is domain, x is a list of attributes and signatures. The result is an augmented SL, if d has x, 'failed otherwise.

(hasAtt p654)
(hasSig p653)
(keyedSystemError p??)

— defun hasAttSig —

(defun |hasAttSig| (d x sl)
  (loop for y in x
until (eq sl 'failed)
do
(setq sl
  (cond
   ((and (consp y) (eq (qcar y) 'attribute)
      (consp (qcdr y)) (eq (qcddr y) nil))
    (hasAtt d (qcadr y) sl))
   ((and (consp y) (eq (qcar y) 'signature)
      (consp (qcdr y)) (consp (qcddr y)) (eq (qcdddr y) nil))
    (hasSig d (qcadr y) (qcaddr y) sl))
   (t
    (keyedSystemError
     "Unexpected error or improper call to system function %1: %2"
     (list "hasAttSig" "unexpected form of unnamed category")))))))

defun hasCate1
[hasCate p663]
[$domPvar p274]
— defun hasCate1 —

(defun |hasCate1| (dom cat sl |$domPvar|)
 (let (|$domPvar|)
  (declare (special |$domPvar|))
  (setq |$domPvar| domPvar)
  (hasCate dom cat sl)))

— defun hasCatExpression —

defun hasCatExpression
[hasCatExpression p657]
[hasCate p663]
[keyedSystemError p??]
— defun hasCatExpression —

(defun |hasCatExpression| (cond sl)
  (let (y)
    (cond
     ((and (consp cond) (eq (qcar cond) 'or))
      (setq sl...
(when
  (let (result)
    (loop for x in (qcdr cond)
      do (setq result
        (or result
          (not (eq (setq y (|hasCatExpression| x sl)) '|failed|))))
      result)
    y))
((and (consp cond) (eq (qcar cond) 'and))
 (when
   (let ((result t))
     (loop for x in (qcdr cond)
       do (setq result
         (and result
           (not (eq (setq sl (|hasCatExpression| x sl)) '|failed|))))
       result)
     sl))
((and (consp cond) (eq (qcar cond) '|has|)
   (consp (qcdr cond)) (consp (qcddr cond)) (eq (qcdddr cond) nil))
 (|hasCate| (qcadr cond) (qcaddr cond) sl))
(t
  (|keyedSystemError|
  "Unexpected error or improper call to system function %1: %2"
  (list "hasSig" "unexpected condition for attribute"))))

defun unifyStruct

[isPatternVar p661]
[unifyStructVar p659]
[unifyStruct p658]

— defun unifyStruct —

(defun unifyStruct| (s1 s2 sl)
 (declare (special |$domPvar| |$hope| |$Coerce| |$Subst|))
 (cond
   ((equal s1 s2) sl)
   (t
    (when (and (consp s1) (eq (qcar s1) '[:])
      (consp (qcdr s1)) (consp (qcddr s1)) (eq (qcdddr s1) nil))
      (setq s1 (qcadr s1)))
    (when (and (consp s2) (eq (qcar s2) '[:])
      (consp (qcdr s2)) (consp (qcddr s2)) (eq (qcdddr s2) nil))
      (setq s2 (qcadr s2)))
    (when (and (null (atom s1)) (eq (car s1) '[:]))
      (setq s1 (qcadr s1)))
    (when (and (null (atom s2)) (eq (car s2) '[:]))
      (setq s2 (qcadr s2))))
(setq s1 (length (cadr s1)))
(when (and (null (atom s2)) (eq (car s2) '|#|))
  (setq s2 (length (cadr s2))))
(cond
  ((equal s1 s2) sl)
  ((|isPatternVar| s1) (|unifyStructVar| s1 s2 sl))
  ((|isPatternVar| s2) (|unifyStructVar| s2 s1 sl))
  ((or (atom s1) (atom s2)) '|failed|)
  (t
   (loop until (or (null s1) (null s2) (eq sl '|failed|))
     do
     (setq sl (|unifyStruct| (car s1) (car s2) sl))
     (setq s1 (cdr s1))
     (setq s2 (cdr s2)))
   (if (or s1 s2) '|failed| sl))))

---

defun unifyStructVar

The first argument is a pattern variable, which is not substituted by sl [contained p??]
[lassoc p??]
[unifyStruct p658]
[constructor? p??]
[subCopy p??]
[containsVars p661]
[canCoerce p??]
[resolveTT p??]
[isPatternVar p661]
[augmentSub p??]
[$domPvar p274]
[$Coerce p??]
[$Subst p??]
[$hope p??]

— defun unifyStructVar —

(defun |unifyStructVar| (v ss sl)
  (let (ps s1 s0 s ns0 ns1 s3)
    (declare (special |$domPvar| |$hope| |$Coerce| |$Subst|))
    (cond
      ((contained v ss) '|failed|)
      (t
        (setq ps (lassoc ss s1))
        (setq s1 (if ps ps ss))
        (cond
          )
    )
  )
  )
((or (setq s0 (assoc v s1)) (setq s0 (assoc v |$Subst|)))
 (setq s (unifyStruct s0 s1 (copy s1)))
 (cond
  ((eq s 'failed)
   (cond
    (and |$Coerce| (null (atom s0)) (constructor? (car s0))
     (cond
      ((or (containsVars s0) (containsVars s1))
       (setq ns0 (subCopy s0 sl))
       (setq ns1 (subCopy s1 sl))
       (cond
        ((or (containsVars ns0) (containsVars ns1))
         (setq |$hope| t)
         'failed)
        (t
         (cond
          (canCoerce ns0 ns1 (setq s3 s1))
          (canCoerce ns1 ns0 (setq s3 s0))
          (t (setq s3 nil)))
          (s3
           (cond
            (not (equal s3 s0))
            (setq s1 (augmentSub v s3 s1)))
           (cond
            ((and (not (equal s3 s1)) (isPatternVar ss))
             (setq s1 (augmentSub ss s3 s1)))
            s1)
         (t 'failed))))))
  (|$domPvar|
   (setq s3 (resolveTT s0 s1))
   (cond
    (s3
     (cond
      (not (equal s3 s0))
      (setq s1 (augmentSub v s3 s1)))
      (cond
       ((and (not (equal s3 s1)) (isPatternVar ss))
        (setq s1 (augmentSub ss s3 s1)))
        s1)
     (t 'failed)))))
  (t 'failed))))
  (t 'failed))))
  (t (augmentSub v ss s)))
  (t (augmentSub v ss s1))))))
defun containsVars

The function containsVars tests whether term t contains a * variable.
[isPatternVar p661]
[containsVars1 p661]

— defun containsVars —

(defun |containsVars| (arg)
  (if (atom arg)
      (|isPatternVar| arg)
      (|containsVars1| arg)))

———

defun isPatternVar

— defun isPatternVar —

(defun |isPatternVar| (v)
  (and (identp v)
       (member v
          '(** *1 *2 *3 *4 *5 *6 *7 *8 *9 *10 *11 *12 *13 *14 *15
             *16 *17 *18 *19 *20))
    t))

———

defun containsVars1

The function containsVars1 tests whether term t contains a * variable. This is a recursive version, which works on a list.
[isPatternVar p661]
[containsVars1 p661]

— defun containsVars1 —

(defun |containsVars1| (arg)
  (let ((t1 (car arg)) (t2 (cdr arg)))
    (if (atom t1)
        (or (|isPatternVar| t1)
            (if (atom t2) ((|isPatternVar| t2) (|containsVars1| t2))
              (or (|containsVars1| t1))))
        (if (atom t2)
            (|containsVars1| t2)
            (or (|containsVars1| t1)
                (|containsVars1| t2))))
(if (atom t2) (|isPatternVar| t2) (|containsVars1| t2))))

---

**defun hasCaty1**

The cond is either a (has a b) or an OR clause of such conditions. SL is augmented, if cond
is true, otherwise the result is ‘failed

[hasCate p663]
[hasCaty1 p662]
[keyedSystemError p??]
[$domPvar p274]

--- defun hasCaty1 ---

(defun |hasCaty1| (cond sl)
  (let (|$domPvar| a s)
    (declare (special |$domPvar|))
    (setq |$domPvar| nil)
    (cond
      ((and (consp cond) (eq (qcar cond) '|has|)
          (consp (qcdr cond)) (consp (qcddr cond)) (eq (qcdddr cond) nil))
       (|hasCate| (qcadr cond) (qcaddr cond) sl))
      ((and (consp cond) (EQ (qcar cond) 'and))
       (loop for x in (qcdr cond)
         while (not (eq s '|failed|))
         do
           (setq s
             (cond
              ((and (consp x) (eq (qcar x) '|has|)
                 (consp (qcdr x)) (consp (qcddr x)) (eq (qcdddr x) nil))
               (|hasCate| (qcadr x) (qcaddr x) sl))
              ((and (consp x) (eq (qcdr x) nil)
                 (consp (qcar x)) (eq (qcaar x) '|has|)
                 (consp (qcdar x)) (consp (qcddar x))
                 (eq (qcdr (qcddar x)) nil))
               (|hasCate| a (qcddar x) sl))
              (t (|hasCaty1| x sl))))))
      ((and (consp cond) (eq (qcar cond) '|or|)
        (do ((next (qcdr cond) (cdr next)) (x nil)
             (nextitem nil (null (eq s '|failed|))))
            ((or (atom next)
              (progn (setq x (car next)) nil)
              nextitem)
             nil)
         (setq s
             (cond
              ((and (consp x) (eq (qcar x) '|or|)
                (consp (qcdr x)) (eq (qcddr x) nil))
               (|hasCate| (qcadr x) (qcaddr x) sl))
              ((and (consp x) (eq (qcdr x) nil)
                (consp (qcar x)) (eq (qcaar x) '|has|)
                (consp (qcdar x)) (consp (qcddar x))
                (eq (qcdr (qcddar x)) nil))
               (|hasCate| a (qcddar x) sl))
              (t (|hasCaty1| x sl)))))
      (t 'failed)))
    s)
(cond
  ((and (consp x) (eq (qcar x) '|has|)
      (consp (qcdr x)) (consp (qcddr x)) (eq (qcdddr x) nil))
    (|hasCate| (qcadr x) (qcaddr x) (copy sl)))))
  ((and (consp x) (eq (qcdr x) nil) (consp (qcar x))
      (eq (qcaar x) '|has|) (consp (qcdar x)) (consp (qcddar x))
      (eq (qcdddar x) nil))
    (|hasCate| (qcadar x) (qcaddar x) (copy sl)))
  (t (|hasCaty1| x (copy sl)))))))

(keyedSystemError| "Unexpected error or improper call to system function %1: %2"
  (list "hasCaty1" "unexpected condition from category table"))))))

---

defun mkDomPvar

[domArg p652]
[length p??]
[$FormalMapVariableList p??]

--- defun mkDomPvar ---

(defun |mkDomPvar| (p d subs y)
  (let (l)
    (declare (special |$FormalMapVariableList|))
    (if (setq l (member p |$FormalMapVariableList|))
      (ldomArg d (- (#| |$FormalMapVariableList|) (#| l) subs y) d))))

---

defun hasCate

[isPatternVar p661]
[hasCate1 p657]
[hasCateSpecial p665]
[containsVariables p??]
[subCopy p??]
[hasCaty p650]
[$EmptyMode p640]
[$Subst p??]
---

### defun hasCate

```lisp
(defun hasCate (dom cat sl)
  (let (ns1 p s sl1)
    (declare (special $hope $Subst $EmptyMode))
    (cond
      ((equal dom $EmptyMode) nil)
      ((isPatternVar dom)
       (cond
        ((and (setq p (assq dom sl))
          (not (eq (setq ns1 (hasCate (cdr p) cat sl)) '|failed|)))
         ns1)
        ((or (setq p (assq dom $Subst)) (setq p (assq dom sl)))
         (setq s (hasCate1 (cdr p) cat sl dom))
         (cond
          ((null (eq s '|failed|)) s)
          (t (hasCateSpecial dom (cdr p) cat sl)))
        (t
         (when (not (eq sl '|failed|)) (setq $hope t))
         '|failed|)))
    (t
     (setq sl1
      (loop for item in sl
        when (null (containsVariables (cdr item)))
        collect item)
     (when sl1 (setq cat (subCopy cat sl1)))
     (hasCaty dom cat sl)))))
```

---

### defun constructSubst

```lisp
(defun constructSubst (d)
  (let (sl (i 0))
    (setq sl (list (cons '#' d)))
    (when (listp d)
      (dolist (x (cdr d))
        (setq i (1+ i))
        (setq sl (cons (cons (internl "#" (princ-to-string i)) x) sl))))
    sl))
```
defun hasCateSpecial

The variable v is a pattern variable, dom is its binding under $\$Subst$. We try to change dom so that it has category cat under sl. The result is a substitution list or 'failed.

(eqcar p??)  
isSubDomain p??  
canCoerceFrom p??  
containsVars p661  
augmentSub p??  
hasCate p663  
hasCaty p650  
hasCateSpecialNew p666  
$\$Integer$ p643  
$\$QuotientField$ p645

— defun hasCateSpecial —

(defun |hasCateSpecial| (v dom cat sl)
  (let (arg d domp nsl)
    (declare (special |$\$Integer| |$\$QuotientField|))
    (cond
      ((and (consp dom) (eq (qcar dom) '|FactoredForm|)
          (consp (qcdr dom)) (eq (qcddr dom) nil))
       (setq arg (qcadr dom))
       (when (isSubDomain arg |$\$Integer|) (setq arg |$\$Integer|))
       (setq d (list '|FactoredRing| arg))
       (setq sl (hasCate arg '(|Ring|) (augmentSub v d sl)))
       (if (eq sl '|failed|) 'failed)
       (hasCaty d cat sl)))
    ((or (eqcar cat '|Field|) (eqcar cat '|DivisionRing|))
     (when (isSubDomain dom |$\$Integer|) (setq dom |$\$Integer|))
     (setq d (list |$\$QuotientField| dom))
     (hasCaty dom '|IntegralDomain| (augmentSub v d sl)))
    ((and (consp cat) (eq (qcar cat) '|PolynomialCategory|)
          (consp (qcdr cat)))
     (setq domp (cons '|Polynomial| (list (qcadr cat))))
     (and (or (containsVars (qcadr cat)) (canCoerceFrom dom domp))
          (hasCaty dom cat (augmentSub v domp sl)))
     (isSubDomain dom |$\$Integer|)
     (setq nsl (hasCate |$\$Integer| cat (augmentSub v |$\$Integer| sl)))
     (if (eq nsl '|failed|) (hasCateSpecialNew v dom cat sl)
         (hasCaty |$\$Integer| cat nsl))
    (t
2
CHAPTER 23. FUNCTION SELECTION

(defun hasCateSpecialNew (v dom cat sl))

(defun hasCateSpecialNew (v dom cat sl)
  (let (fe alg fefull d partialResult)
    (declare (special |$RationalNumber| |$ComplexInteger| |$Integer|))
    (setq fe
      (|member| (qcar cat)
        '([ElementaryFunctionCategory]
          |TrigonometricFunctionCategory|
          |ArcTrigonometricFunctionCategory|
          |HyperbolicFunctionCategory|
          |ArcHyperbolicFunctionCategory|
          |PrimitiveFunctionCategory|
          |SpecialFunctionCategory|
          |Evalable|
          |CombinatorialOpsCategory|
          |TranscendentalFunctionCategory|
          |AlgebraicallyClosedFunctionSpace|
          |ExpressionSpace|
          |LiouvillianFunctionCategory|
          |FunctionSpace|)))
    (setq alg
      (|member| (qcar cat)
        '([RadicalCategory]
          |AlgebraicallyClosedField|)))
    (setq fefull
      (or fe alg (eqcar cat '|CombinatorialFunctionCategory|)))
    (setq partialResult
      (cond
        ((or (eqcar dom '|Variable|) (eqcar dom '|Symbol|))))
(cond
  ((|member| (car cat)
    '|SemiGroup|
    |AbelianSemiGroup|
    |Monoid|
    |AbelianGroup|
    |AbelianMonoid|
    |PartialDifferentialRing|
    |Ring|
    |InputForm|)
   (setq d (list '|Polynomial| |$Integer|)))
  (|augmentSub| v d sl))
  ((eqcar cat '|Group|)
   (setq d (list '|Fraction| (list '|Polynomial| |$Integer|)))
   (|augmentSub| v d sl))
  (fefull
   (setq d (|defaultTargetFE| dom))
   (|augmentSub| v d sl))
  (t '|failed|)))
((|isEqualOrSubDomain| dom |$Integer|)
 (cond
  (fe
   (setq d (|defaultTargetFE| |$Integer|))
   (|augmentSub| v d sl))
  (alg
   (setq d '(|AlgebraicNumber|))
   (|augmentSub| v d sl))
  (t '|failed|)))
((equal (|underDomainOf| dom) |$ComplexInteger|)
 (setq d (|defaultTargetFE| |$ComplexInteger|))
 (|hasCaty| d cat (|augmentSub| v d sl)))
((and (equal dom |$RationalNumber|) alg)
 (setq d '(|AlgebraicNumber|))
 (|augmentSub| v d sl))
(fefull
 (setq d (|defaultTargetFE| dom))
 (|augmentSub| v d sl))
 (t '|failed|)))
(if (eq partialResult '|failed|)
  '|failed|
  (|hasCaty| d cat partialResult))))

---

defun defaultTargetFE
defun defaultTargetFE ...

(defun defaultTargetFE (a &rest (options nil))
  (let ((a (car dom))
        (options (cdr dom)))
    (cond
      ((or (consp a) (eq (qcar a) '|Variable|)
        (consp (qcdr a)) (eq (qcddr a) nil))
       (equal a |$RationalNumber|)
       (member (qcar a) (list (qcar |$Symbol|) '|RationalRadicals| '|Pi|)))
      (equal a |$SingleInteger|)
      (isEqualOrSubDomain a |$Integer|)
      (equal a '(|AlgebraicNumber|)))
      (if (ifcar options)
        (list |$FunctionalExpression| (list '|Complex| |$Integer|))
        (list |$FunctionalExpression| |$Integer|)))
    ((and (consp a) (eq (qcar a) '|Complex|)
        (consp (qcdr a)) (eq (qcddr a) nil))
     (defaultTargetFE (qcadr a) t))
    ((and (consp a) (consp (qcdr a)) (eq (qcddr a) nil)
        (member (qcar a) '(|Polynomial| |RationalFunction| |Fraction|)))
     (defaultTargetFE (qcadr a) (ifcar options)))
    ((and (consp a) (equal (qcar a) |$FunctionalExpression|)
        (consp (qcdr a)) (eq (qcddr a) nil))
     a)
    ((ifcar options)
     (list |$FunctionalExpression| (list '|Complex| a)))
    (t
     (list |$FunctionalExpression| a))))

——-

defun isEqualOrSubDomain ...

defun isEqualOrSubDomain (d1 d2)
(or (equal d1 d2)
  (|isSubDomain| d1 d2)
  (and (atom d1)
       (or (and (consp d2) (eq (qcar d2) '|Variable|)
                (consp (qcdr d2))
                (eq (qcddr d2) nil)
                (equal (qcadr d2) d1))
       (and (consp d2) (eq (qcdr d2) nil)
                (equal (qcar d2) d1)))))

(and (atom d2)
     (or (and (consp d1) (eq (qcar d1) '|Variable|)
           (consp (qcdr d1)) (eq (qcddr d1) nil)
           (equal (qcadr d1) d2))
     (and (consp d1) (eq (qcdr d1) nil)
           (equal (qcar d1) d2))))))
Chapter 24

Coercions

main algorithms for canCoerceFrom and coerceInteractive

canCoerceFrom and coerceInteractive are the two coercion functions
for $InteractiveMode. They translate RN, RF and RR to QF I, QF P
and RE RN, respectively, and call coerceInt or canCoerce, which
both work in the same way (e.g. coercion from t1 to t2):
1. they try to coerce t1 to t2 directly (tower coercion), and, if
   this fails, to coerce t1 to the last argument of t2 and embed
   this last argument into t2. These embedding functions are now only
defined in the algebra code. (RSS 2-27-87)
2. the tower coercion looks whether there is any applicable local
   coercion, which means, one defined in boot or in algebra code.
   If there is an applicable function from a constructor, which is
   inside the type tower of t1, to the top level constructor of t2,
   then this constructor is bubbled up inside t1. This means,
special coercion functions (defined in boot) are called, which
   commute two constructors in a tower. Then the local coercion is
called on these constructors, which both are on top level now.
example:
let t1 = A B C D E (short for (A (B (C (D (E))))), where A ... E are
   type constructors), and t2 = F D G H I J
there is no coercion from t1 to t2 directly, so we try to coerce
   t1 to s1 = D G H I J, the last argument of t2
we create the type s2 = A D B C E and call a local coercion A2A
   from t1 to s2, which, by recursively calling coerce, bubbles up
the constructor D
then we call a commute coerce from s2 to s3 = D A B C E and a local
   coerce D2D from s3 to s1
finally we embed s1 into t2, which completes the coercion t1 to t2
the result of canCoerceFrom is TRUE or NIL
the result of coerceInteractive is a object or NIL (=failed)
all boot coercion functions have the following result:
1. if $u=$fromCoerceable$, then TRUE or NIL
2. if the coercion succeeds, the coerced value (this may be NIL)
3. if the coercion fails, they throw to a catch point in
   coerceByFunction

defun coerceInteractive

[|objMode p448|]
[|objVal p448|]
[|clearDependentMaps p??|]
[|throwKeyedMsg p??|]
[|startTimingProcess p??|]
[|mkObj p146|]
[|mkObjWrap p447|]
[|coerceInt0 p674|]
[|stopTimingProcess p??|]
[|insideCoerceInteractive p??|]
[|$OutputForm p642|]
[|$mapName p??|]
[|$compilingMap p274|]
[|$NoValueMode p??|]
[|$EmptyMode p640|]

— defun coerceInteractive —

(defun |coerceInteractive| (triple t2)
  (let ((|$insideCoerceInteractive| t1 val expr2 result)
        (declare (special |$insideCoerceInteractive| |$OutputForm|
                   |$mapName| |$compilingMap| |$NoValueMode| |$EmptyMode|))
    (setq t1 (|objMode| triple))
    (setq val (|objVal| triple))
    (cond
      ((or (null t2) (equal t2 |$EmptyMode|)) nil)
      ((equal t2 t1) triple)
      ((equal t2 '|$NoValueMode|) (mkObj val t2))
      (t
       (when (eq (car t2) '|SubDomain|) (setq t2 (second t2)))
       (cond
        ((|member| t1 t1"]).(\{Category\} \{Mode\} \{Domain\} \{SubDomain\} \{Domain\}))
        (when (equal t2 |$OutputForm|) (mkObj val t2)))
        ((equal t1 '|$NoValueMode|)
         (when |$compilingMap| (|clearDependentMaps| |$mapName| nil))
         (|throwKeyedMsg|)
         (format nil
               "You are trying to use something (probably a loop) in a ")
         (stopTimingProcess)))
    (throwKeyedMsg)
    (stopTimingProcess)
    (return (coerceInt0 result))
  (return nil)))
situation where a value is expected. In particular, you are trying to convert this to the type \%1p. The following information may help: possible function name: %2p

(list t2 |$mapName|)

(t
(setq |$insideCoerceInteractive| t)
(setq expr2 (equal t2 |$OutputForm|))
(cond
  (expr2 (|startTimingProcess| '|print|))
  (t (|startTimingProcess| '|coercion|)))
(setq result
  (cond
    ((and expr2 (equal t1 val)) (mkObj val |$OutputForm|))
    ((and expr2 (eq (car t1) '|Variable|))
     (mkObjWrap (second t1) |$OutputForm|))
    (t (|coerceInt0| triple t2))))
  (cond
    (expr2 (|stopTimingProcess| '|print|))
    (t (|stopTimingProcess| '|coercion|)
result)))))))

defun coerceInt

defun |coerceInt| (triple t2)
(let (val newMode newVal)
  (if (setq val (|coerceInt1| triple t2))
    val
    (when (eq (car (|objMode| triple)) '|Variable|)
      (setq newMode (|getMinimalVarMode| (|unwrap| (|objVal| triple)) nil))
      (setq newVal (|coerceInt| triple newMode))
      (|coerceInt| newVal t2))))

defun coerceInt
defun coerceInt0

[defun coerceInt0]

This is the top level interactive coercion, which transfers all RN, RF and RR into equivalent types

— defun coerceInt0 —

(defun coerceInt0 (triple t2)
  (prog (val t1 s1 s2 let1 t1p valp ans x)
    (declare (special $OutputForm $Any $genValue))
    (return
      (progn
        (setq val (objVal triple))
        (setq t1 (objMode triple))
        (cond
          ((eq val '$fromCoerceable) (canCoerceFrom t1 t2))
          ((equal t1 t2) triple)
          (t
            (cond
              ((equal t2 $OutputForm) (setq s1 t1) (setq s2 t2))
              (t
                (setq s1 t1)
                (setq s2 t2)
                (when (equal s1 s2) (return (mkObj val t2))))))))
    (cond
      ; handle case where we must generate code
      ((and (null (isWrapped val))
        (or
          (null (eq (car t1) 'FunctionCalled))
          (null $genValue)))))
    (intCodeGenCOERCE triple t2))
  (and (equal t1 $Any)
    (nequal t2 $OutputForm)
    (progn
      (setq let1 (unwrap val))
      (setq tip (car let1)))
  )
(setq valp (cdr let1))
(let1)
(setq ans (|coerceInt0| (mkObjWrap valp t1p) t2)))
(ans)
(t
(unless (eq s1 t1) (setq triple (mkObj val s1))))
(when (setq x (|coerceInt| triple s2))
(cond
((eq s2 t2) x)
(t
(|objSetMode| x t2)
x)))))))))))

defun coerceInt1

This is general interactive coercion. The result is a new triple with type m2 or NIL (= failed). [NRTcompileEvalForm p??][absolutelyCanCoerceByCheating p702][asTupleAsList p??][bottomUp p??][coerceByFunction p680][coerceInt1 p675][coerceInt2Union p697][coerceIntAlgebraicConstant p698][coerceIntFromUnion p697][coerceIntTower p682][coerceIntX p700][coerceInt p673][coerceRetract p709][coerceSubDomain p700][compareTypeLists p700][deconstructT p??][evalDomain p1075][getFunctionFromDomain p??][getValue p??][isEqualOrSubDomain p668][isSubDomain p??][mkAtreeNode p??][mkAtree p??][mkObjWrap p447][mkObj p446][nequal p??][nreverse0 p??]
defun coerceInt1 (triple t2)
  (prog ($useCoerceOrCroak t1 sentinel t1p valp s body vars tree val symNode mms ml oldName intName t3 triplep let1 arg tt ans)
    (declare (special $useCoerceOrCroak $Integer $QuotientField $e $genValue $Symbol $AnonymousFunction $OutputForm $String $Any $Void $SingleInteger $NonNegativeInteger $PositiveInteger $EmptyMode))
    (return
      (seq
        (progn
          (setq $useCoerceOrCroak t)
          (cond
            ((equal t2 $EmptyMode) nil)
            (t
              (setq t1 (|objMode| triple))
              (cond
                ((equal t1 t2) triple)
                (t
                  (setq val (|objVal| triple))
                  (cond
                    ((absolutelyCanCoerceByCheating t1 t2) (mkObj val t2))
                    ((isSubDomain t2 t1) (|coerceSubDomain| val t1 t2))
                    (t
                      nil)))))))))
(cond
  ((equal t1 |$SingleInteger|)
    (cond
      ((or (equal t2 |$Integer|) (equal t2 |$SingleInteger|))
       (return (mkObj val t2)))
      (t
       (setq sintp (typep val 'fixnum))
       (cond
        ((and sintp (equal t2 |$PositiveInteger|) (> val 0))
         (return (mkObj val t2)))
        ((and sintp (equal t2 |$NonNegativeInteger|) (>= val 0))
         (return (mkObj val t2))))))))
  (cond
    ((and (equal t2 |$SingleInteger|) (isEqualOrSubDomain t1 |$Integer|) (integerp val))
     (cond
      ((typep val 'fixnum) (mkObj val t2))
      (t nil)))
    ((equal t2 |$Void|) (mkObj (|voidValue|) |$Void|))
    ((equal t2 |$Any|) (mkObjWrap (cons t1 (|unwrap| val)) '(|Any|)))
    ((and (equal t1 |$Any|) (nequal t2 |$OutputForm|) (progn
      (setq let1 (|unwrap| val))
      (setq t1p (car let1))
      (setq valp (cdr let1))
      let1)
     (setq ans (|coerceInt| (mkObjWrap valp t1p) t2)))
     ans)
    ; tagged union selectors
    ((or (and (eq (car t1) '|Variable|) (equal (cadr t1) t2))
         (and (eq (car t2) '|Variable|) (equal (cadr t2) t1))
         (mkObj val t2))
     (stringp t2)
     (cond
      ((and (eq (first t1) '|Variable|)
              (equal t2 (pname (second t1))))
       (mkObjWrap t2 t2))
      (t
       (setq valp (|unwrap| val))
       (when (and (equal t2 valp)
                  (or (equal valp t1) (equal t1 |$String|))
                  (mkObj val t2)))))))
  (eq (first t1) '|Tuple|)
  (|coerceInt1|)
  (mkObjWrap
   (|asTupleAsList| (|unwrap| val))
   (list '|List| (setq s (second t1)))
   t2))
((and (consp t1) (eq (qcar t1) '|Union|))
 (|coerceIntFromUnion| triple t2))
((and (consp t2) (eq (qcar t2) '|Union|))
 (|coerceInt2Union| triple t2))
((and (stringp t1) (equal t2 |$String|))
 (mkObj val |$String|))
((and (stringp t1) (eq (car t2) '|Variable|))
 (when (equal t1 (pname (second t2))) (mkObjWrap (second t2) t2)))
((and (stringp t1) (equal t1 (|unwrap| val)))
 (when (equal t2 |$OutputForm|) (mkObj t1 |$OutputForm|))))
((atom t1) nil)
(t
 (cond
 ((and (equal t1 |$AnonymousFunction|)
    (eq (car t2) '|Mapping|))
 (setq |$useCoerceOrCroak| nil)
 (setq let1 (|unwrap| val))
 (setq vars (cadr let1))
 (setq body (cddr let1))
 (setq vars
  (cond
   ((atom vars) (cons vars nil))
   ((and (consp vars) (eq (qcar vars) '|Tuple|)) (cdr vars))
   (t vars)))
 (cond
 ((nequal (|#| (cddr t2)) (|#| vars)) '|continue|)
 (t
 (setq tree
  (|mkAtree|
   (cons 'adef
    (cons vars
     (cons (cons (cadr t2) (cddr t2))
      (cons (loop for x in (cdr t2) collect nil)
       body)))))))))
 (cond
 ((eq
   (catch '|coerceOrCroaker| (|bottomUp| tree) '|croaked|) nil)
   (t (return (|getValue| tree)))))))
 (cond
 ((and (equal t1 |$Symbol|) (eq (car t2) '|Mapping|))
  (cond
   ((null (setq mms
    (|selectMms1| (|unwrap| val) nil
     (cddr t2) (cddr t2) (cadr t2))))
    nil)
   (t
    (cond
     ((nequal (cadaar mms) (cadr t2)) nil)
     (|$genValue|...)))))
(mkObjWrap
 (getFunctionFromDomain
  (unwrap val) (caaar mms) (cddaar mms)) t2))

(t
 (mkObj
  (NRTcompileEvalForm
   (unwrap val) (cdaar mms) (evalDomain (caaar mms))
   t2)))))

((and (eq (car t1) '|Variable|) (eq (car t2) '|Mapping|))
 (setq mms
  (selectMms1 (cadr t1) (cadr t2) (cddr t2) (cddr t2) nil))
 (cond
   ((and (null mms)
         (null
          (setq mms
            (selectMms1 (cadr t1) (cadr t2)
                         (cddr t2) (cddr t2) t)))))
    nil)
   (t
    (cond
     ((nequal (cadaar mms) (cadr t2)) nil)
     ((eq (caaaar mms) '|_FreeFunction_|)
      (mkObj (cdaaar mms) t2))
     ($genValue|
      (mkObjWrap
       (getFunctionFromDomain (cadr t1) (caaar mms)
                                (cddaar mms)) t2))
    (t
     (mkObj
      (NRTcompileEvalForm (cadr t1) (cdr (caar mms))
                           (evalDomain (caaar mms))
                           t2)))))))

((and (eq (car t1) '|FunctionCalled|) (eq (qcar t2) '|Mapping|))
 (setq symNode (mkAtreeNode (cadr t1)))
 (transferPropsToNode (cadr t1) symNode)
 (cond
   ((null
     (setq mms
      (selectLocalMms symNode (cadr t1) (cddr t2) (cadr t2)))))
    nil)
   (t
    (cond
     ((nequal (cadaar mms) (cadr t2)) nil)
     (t
      (setq ml (cons (cadr t2) (cddr t2)))
      (setq intName
       (when
        (some #'(lambda (mm)
                     (setq oldName (second mm))
                     (compareTypeLists (cdr mm) ml) mms))
       t)
      (setq m2 (cons (cadr t2) (cddr t2)))
      )))
(cond
  ((null intName) nil)
  (t (mkObjWrap intName t2))))
((eq (car t1) '|FunctionCalled|)
  (setq t3 (|get| (second t1) '|model| |$e|)))
  (when (and (eq (car t3) '|Mapping|)
               (setq triplep ([coerceInt| triple t3])))
    ([coerceInt| triplep t2)])
  ((and (eq (car t1) '|Variable|)
        (consp t2)
        (or (|isEqualOrSubDomain| t2 |$Integer|)
            (equal t2 (list |$QuotientField| |$Integer|))
            (member (car t2)
                '(|RationalNumber| |BigFloat| |NewFloat| |Float| |DoubleFloat|)))
            nil)
    (t
     (setq ans
       (or
        ([coerceRetract| triple t2]
         ([coerceIntTower| triple t2]
          (progn
            (setq arg (cdr ([deconstructT| t2])))
            (and arg
              (progn
                (setq tt ([coerceInt| triple (|last| arg)])
                  (and tt ([coerceByFunction| tt t2]))))))
            (or ans
              (and (|isSubDomain| t1 |$Integer|)
                ([coerceInt| (mkObj val |$Integer|) t2])
                ([coerceIntAlgebraicConstant| triple t2]
                 ([coerceIntX| val t1 t2])))))))))))))

---

(defun coerceByFunction
  "---
---

(defun coerceByFunction| (t$ m2)
  (let ($ m1 ud x tmp1 a tmp2 b funName mm dc tar args slot dcVector fun fn
d val env code m1p m2p)
    (declare (special $ |$coerceFailure| |$Boolean|))
    (setq x (|objVal| T$))
    (cond
      ((eq x '|$fromCoerceable$|) nil)
((eq (car m2) '|Union|) nil)
(t
(setq m1 (|objMode| t$))
(cond
  ((and (consp m2) (eq (qcar m2) '|Boolean|
               (consp m1) (eq (qcar m1) '|Equation|)
          (PROGN
            (setq tmp1 (cdr m1))
            (and (consp tmp1) (eq (cdr tmp1) nil)
             (progn (setq ud (car tmp1)) t)))))))
(setq dcVector (evalDomain| ud))
(setq fun
  (cond
   (isWrapped| x)
   (NRTcompiledLookup| '= (list |$Boolean| '$ '$) dcVector))
   (t
    (NRTcompileEvalForm| '= (list |$Boolean| '$ '$) dcVector)))))
(setq fn (car fun))
(setq d (cdr fun))
(setq x (|unwrap| x))
(mkObj|Wrap| (spadcall (car x) (cdr x) fun) m2))
((null (and (consp x) (eq (car x) '|spadcall|
          (progn
            (setq tmp1 (cdr x))
            (and (consp tmp1)
             (progn
              (setq a (car tmp1))
              (setq tmp2 (cdr tmp1))
              (and (consp tmp2)
               (progn
                (setq b (car tmp2)) t)))))))
   (keyedSystemError| "Generated code is incorrect for equation" nil))
  (t
   (setq code (list 'spadcall a b fun))
   (mkObj code |$Boolean|))))
(t
  (cond
   (null
    (setq mm (|coerceConvertMmSelection| (setq funName '|coerce|) m1 m2)))
    (setq mm
      (|coerceConvertMmSelection| (setq funName '|convert|) m1 m2))))
  (when mm
    (setq dc (caar mm))
    (setq tar (cadar mm))
    (setq args (cddar mm))
    (setq slot (cdr mm))
    (setq dcVector (evalDomain| dc))
    (setq fun
(cond
  ((|isWrapped| x) (|NRTcompiledLookup| funcName slot dcVector))
  (t (|NRTcompileEvalForm| funcName slot dcVector))))
(setq fn (car fun))
(setq d (cdr fun))
(cond
  ((equal fn #'|Undef|) nil)
  ((|isWrapped| x)
   (setq $ dcVector)
   (setq val (catch '|coerceFailure| (spadcall (|unwrap| x) fun)))
   (cond
    ((equal val |$coerceFailure|) nil)
    (t (mkObjWrap val m2)))
  (t
   (setq env fun)
   (setq code (list '|failCheck| (list 'spadcall x env)))
   (mkObj code m2))))))))

---

defun coerceIntTower

This tries to find a coercion from top level t2 to somewhere inside t1 It builds a new argument type, for which coercion is called recursively [coerceIntPermute p685]
[coerceIntSpecial p692]
[last p??]
[coerceIntTest p684]
[constructT p??]
[replaceLast p??]
[deconstructT p??]
[bubbleConstructor p??]
[isValidType p??]
[coerceIntCommute p689]
[coerceIntByMap p693]
[coerceIntTableOrFunction p690]

--- defun coerceIntTower ---

(defun |coerceIntTower| (triple t2)
  (let (t1 c1 arg1 tt c arg t1 let1 c2 arg2 s x)
    (cond
     ((setq x (|coerceIntByMap| triple t2)) x)
     ((setq x (|coerceIntCommute| triple t2)) x)
     ((setq x (|coerceIntPermute| triple t2)) x)
     ((setq x (|coerceIntSpecial| triple t2)) x)
     ((setq x (|coerceIntTableOrFunction| triple t2)) x))
    (t
     (setq env fun)
     (setq code (list '|failCheck| (list 'spadcall x env)))
     (mkObj code m2)))))))

(setq t1 (objMode triple))
(setq let1 (deconstructT t1))
(setq c1 (car let1))
(setq arg1 (cdr let1))
(and arg1
  (progn
    (setq t1 nil)
    (setq arg arg1)
    (loop until (or x (not arg)) do
      (setq tt (last arg))
      (setq let1 (deconstructT tt))
      (setq c (car let1))
      (setq arg (cdr let1))
      (setq tl (cons c (cons arg tl)))
      (cond
        ((setq x (and arg (coerceIntTest tt t2)))
          (cond
            ((cddr t1)
              (setq s
                (constructT c1
                  (replaceLast arg1 (bubbleConstructor t1))))
              (cond
                ((null (isValidType s)) (setq x nil))
                ((setq x (or (coerceIntByMap triple s)
                    (coerceIntTableOrFunction triple s)))
                  (setq let1 (deconstructT (last s)))
                  (setq c2 (car let1))
                  (setq arg2 (cdr let1))
                  (setq s (bubbleConstructor (list c2 arg2 c1 arg1)))
                  (cond
                    ((null (isValidType s)) (setq x nil))
                    ((setq x (coerceIntCommute x s))
                      (setq x (or (coerceIntByMap x t2)
                        (coerceIntTableOrFunction x t2))))))))
        (t
          (setq s (bubbleConstructor (list c arg c1 arg1)))
          (cond
            ((null (isValidType s)) (setq x nil))
            ((setq x (coerceIntCommute triple s))
              (setq x (or (coerceIntByMap x t2)
                (coerceIntTableOrFunction x t2)))))))))))

defun coerceIntTest
This looks whether there exists a table entry or a coercion function. Thus the type can be
bubbled before coerceIntTableOrFunction is called. [coerceConvertMmSelection p684]

---

(defun coerceIntTest (t1 t2)
  (let (p b)
    (declare (special $useConvertForCoercions $CoerceTable))
    (or (equal t1 t2)
      (setq b
        (and (setq p (assq (car t1) $CoerceTable))
          (assq (car t2) (cdr p)))
        (or b
          (coerceConvertMmSelection 'coerce t1 t2)
          (and $useConvertForCoercions
            (coerceConvertMmSelection 'convert t1 t2)))))

---

defun coerceConvertMmSelection
This calls selectMms with $Coerce=NIL and tests for required target type. funName is either
'coerce or 'convert.

---

(defun coerceConvertMmSelection (&rest g1)
  (labels ((checktargets (funName m1 m2)
              (let (|$declaredMode| |$reportBottomUpFlag|
                (declare (special |$declaredMode| |$reportBottomUpFlag|)
                  (setq |$declaredMode| nil)
                  (setq |$reportBottomUpFlag| nil)

---
(car
(loop for x in (|selectMms1| funName m2 (list m1) (list m1) nil)
collect
 (when (and (|hasCorrectTarget| m2 (car x)) (equal (caddar x) m1))
 (cons (car x) (cons (cons (cadar x) (list (cadadr x))) (cddr x))))))))
(let (g3)
 (if (setq g3 (hget |coerceConvertMmSelection;AL| g1))
 ([[CDRwithIncrement| g3]
 (cdr (hput |coerceConvertMmSelection;AL| g1
 (cons 1 (apply #'checktargets g1))))))))

defun hasCorrectTarget
This tests whether the target of signature sig is either m or a union containing m. It also
 discards TEQ as it is not meant to be used at top-level
 — defun hasCorrectTarget 0 —

(defun |hasCorrectTarget| (m sig)
 (let (tar)
 (setq tar (second sig))
 (cond
 ((eq (caar sig) '|TypeEquivalence|) nil)
 ((equal m tar) t)
 ((and (eq (car tar) '|Union|)
 (eq (third tar) '|failed|))
 (equal (second tar) m))
 ((and (eq (car tar) '|Union|)
 (eq (second tar) '|failed|)
 (equal (third tar) m))))))

——

defun coerceIntPermute

——

[member p1198]
[objcMode p448]
[computeTTTranspositions p686]
[coerceInt p673]

— defun coerceIntPermute —

(defun |coerceIntPermute| (object t2)
 (let (t1 towers ok)
(cond
  ((|member| t2 '((|Integer|) (|OutputForm|))) nil)
  (t
    (setq t1 (|objMode| object))
    (setq towers (|computeTTTranspositions| t1 t2))
    ; At this point, CAR towers = t1 and last towers should be similar
    ; to t2 in the sense that the components of t1 are in the same order
    ; as in t2. If length towers = 2 and t2 = last towers, we quit to
    ; avoid an infinite loop.
    (cond
      ((or (null towers) (null (cdr towers))) nil)
      ((and (null (cddr towers)) (equal t2 (cadr towers))) nil)
      (t
        (setq ok t)
        ; do the coercions successively, quitting if any fail
        (loop for tt in (cdr towers) while ok do
          (unless (setq object (|coerceInt| object tt)) (setq ok nil)))
        (when ok object))))))

---

defun computeTTTranspositions

[decomposeTypeIntoTower p688]
[member p1198]
[nequal p??]
[msort p??]
[remdup p??]
[length p??]
[list2vec p??]
[permuteToOrder p687]
[setelt p??]
[vec2list p1210]
[reassembleTowerIntoType p689]

— defun computeTTTranspositions —

(defun |computeTTTranspositions| (t1 t2)
  (labels (
    (compress (z start len)
      (cond
        ((>= start len) z)
        ((|member| start z) (compress z (1+ start) len))
        (t
          (compress
            (loop for i in z do collect (if (> start i) i (1- i))) start len))))
    (let (t1 t12 p2p n1 p2 perms tower tt towers)
      (compress z start len)))
      (loop for tt in (cdr towers) while ok do
        (unless (setq object (|coerceInt| object tt)) (setq ok nil)))
      (when ok object))))))

---
; decompose t1 into its tower parts
(setq tl1 (|decomposeTypeIntoTower| t1))
(setq tl2 (|decomposeTypeIntoTower| t2))
(cond
 ; if not at least 2 parts, don't bother working here
((null (and (cdr tl1) (cdr tl2))) nil)
(t
 ; determine the relative order of the parts of t1 in t2
(setq p2 (nreverse0 (loop for d1 in tl1 collect (position d1 tl2))))
(cond
 ; something not present
((|member| (- 1) p2) nil)
(t
 ; if they are all ascending, this function will do nothing
(setq p2p (msort p2))
(cond
 ; if anything is repeated twice, leave
((nequal p2p (msort (remdup p2p))) nil)
(t
 ; create a list of permutations that transform the tower parts
 ; of t1 into the order they are in in t2
(setq n1 (|#| tl1))
(setq p2 (list2vec (compress p2 0 (|#| (remdup tl1)))))
; p2 now has the same position numbers as p1, we need to determine
; a list of permutations that takes p1 into p2.
(setq perms (|permuteToOrder| p2 (- n1 1) 0))
(setq towers (list tl1))
(setq tower (list2vec tl1))
(loop for perm in perms do
  (setq tt (elt tower (car perm)))
  (setelt tower (car perm) (elt tower (cdr perm)))
  (setelt tower (cdr perm) tt)
  (setq towers (cons (vec2list tower) towers)))
(setq towers (nreverse0
  (loop for tower in towers collect (|reassembleTowerIntoType| tower))))
(unless (equal (car towers) t2) (setq towers (cons t2 towers))))
(nreverse towers))))))))

defun permuteToOrder
[permuteToOrder p687]
[setelt p??]

— defun permuteToOrder —
(defun permuteToOrder (p n start)
  (let (r x perms tt stpos)
    (setq r (- n start))
    (cond
      ((<= r 0) nil)
      ((eql r 1)
       (cond
        ((> (elt p (+ r 1)) (elt p r)) nil)
        (t (list (cons r (+ r 1))))))
      ((equal (elt p start) start) (permuteToOrder p n (+ start 1)))
      (t
       (setq stpos nil)
       (loop for i from (+ start 1) to n while (not stpos) do
         (when (equal (elt p i) start) (setq stpos i)))
       (setq perms nil)
       (loop while (not (equal stpos start)) do
         (setq x (- stpos 1))
         (setq perms (cons (cons x stpos) perms))
         (setq tt (elt p stpos))
         (setelt p stpos (elt p x))
         (setelt p x tt)
         (setq stpos x))
       (append (nreverse perms) (permuteToOrder p n (+ start 1)))))))

(defun decomposeTypeIntoTower
  (decomposeTypeIntoTower (tt))
  (cond
   ((atom tt) (list tt))
   ((null (cdr (deconstructT tt))) (list tt))
   (t
    (setq rd (reverse tt))
    (cons (reverse (cdr rd)) (decomposeTypeIntoTower (car rd))))))
defun reassembleTowerIntoType

(defun |reassembleTowerIntoType| (tower)
  (let (let1)
    (cond
      ((atom tower) tower)
      ((null (cdr tower)) (car tower))
      (t
       (setq let1 (reverse tower))
       (|reassembleTowerIntoType|
        (append (nreverse (cddr let1))
           (list (append (second let1) (list (first let1))))))))))

defun coerceIntCommute

(defun |coerceIntCommute| (obj target)
  (let (source s t$ d fun u c)
    (declare (special |$coerceFailure|))
    (setq source (|objMode| obj))
    (cond
      ((null (|coerceCommuteTest| source target)) nil)
      (t
       (setq s (|underDomainOf| source))
       (setq t$ (|underDomainOf| target))
       (cond
        ((equal source t$) nil)
        ((setq d (car source))
         (setq fun


(or (getl d 'coerceCommute))
  (intern (strconc "commute" (princ-to-string d))))
(cond
  ((canFuncall? fun)
   (put d 'coerceCommute fun)
   (setq u (|objValUnwrap| obj))
   (setq c (catch 'coerceFailure (funcall fun u source s target t$)))
   (cond
    ((equal c $coerceFailure) nil)
    ((eq u '$fromCoerceable) c)
    (t (mkObjWrap c target)))))))))

---

defun coerceCommuteTest

[isLegitimateMode p??]
[underDomainOf p??]
[deconstructT p??]

— defun coerceCommuteTest —

(defun |coerceCommuteTest| (t1 t2)
  (let (u1 u2)
    (cond
      ((null (|isLegitimateMode| t2 nil nil)) nil)
      ((null (setq u1 (|underDomainOf| t1))) nil)
      ((null (setq u2 (|underDomainOf| t2))) nil)
      ((null (|underDomainOf| u1)) nil)
      ((null (|underDomainOf| u2)) nil)
      (t
       (and (equal (car (|deconstructT| t1)) (car (|deconstructT| u2)))
            (equal (car (|deconstructT| t2)) (car (|deconstructT| u1)))))))

---

defun coerceIntTableOrFunction

This function does the actual coercion to t2, but not to an argument type of t2 [isValidType p??]
[isLegitimateMode p??]
[objMode p448]
[assq p1200]
[coerceByTable p691]
[objVal p448]
(defun |coerceIntTableOrFunction| (triple t2)
  (let (t1 p tmp1)
    (declare (special |$CoerceTable|))
    (cond
      ((null (|isValidType| t2)) nil)
      ((null (|isLegitimateMode| t2 nil nil)) nil)
      (t
        (setq t1 (|objMode| triple))
        (setq p (assq (car t1) |$CoerceTable|))
        (cond
          ((and p (setq tmp1 (assq (car t2) (cdr p))))
           (cond
            ((eq (third tmp1) '|Identity|) (mkObj (|objVal| triple) t2))
            ((eq (second tmp1) '|total|)
              (or (|coerceByTable| (third tmp1) (|objVal| triple) t1 t2 t)
                  (|coerceByFunction| triple t2)))
            (t (|coerceByFunction| triple t2))))
          (t (|coerceByFunction| triple t2))))
    )))

---

defun coerceByTable

[isWrapped p??] [unwrap p??] [mkObjWrap p447] [isTotalCoerce p??] [mkObj p446] [mkq p??] [$OutputForm p642] [$coerceFailure p??] [coerceFailure p??]

---

defun coerceByTable

(defun |coerceByTable| (fn x t1 t2 isTotalCoerce)
  (let (c)
    (declare (special |$coerceFailure| |$OutputForm|))
    (cond
defun catchCoerceFailure

This function is funcalled from code constructed by coerceByTable. [unwrap p??] [wrap p1193] [throwKeyedMsgCannotCoerceWithValue p??] [$coerceFailure p??] [coerceFailure p??]

— defun catchCoerceFailure —

(defun |catchCoerceFailure| (fn x t1 t2)
 (let (c)
   (declare (special |$coerceFailure|))
   (setq c (catch '|coerceFailure| (funcall fn x t1 t2)))
   (if (equal c |$coerceFailure|)
       (throwKeyedMsgCannotCoerceWithValue (|wrap| (|unwrap| x)) t1 t2)
       c)))

— defun coerceIntSpecial —

defun coerceIntSpecial

[objMode p448] [coerceInt p673]

— defun coerceIntSpecial —

(defun |coerceIntSpecial| (triple t2)
 (let (x)
   (when (and (eq (first t2) '|SimpleAlgebraicExtension|) (equal (second t2) |objMode| triple))
     (unless (setq x (|coerceInt| triple (third t2)))
      (|coerceInt| x t2))))))
defun coerceIntByMap

The idea is this: if \( t_1 \) is \( D \cup U_1 \) and \( t_2 \) is \( D \cup U_2 \), then look for a map: \( (U_1 \rightarrow U_2, D \cup U_1) \rightarrow D \cup U_2 \). If it exists, then create a function to do the coercion on the element level and call the map function. [objMode p448]

| \( \text{length p??} \) |
| \( \text{deconstructT p??} \) |
| \( \text{nequal p??} \) |
| \( \text{valueArgsEqual? p695} \) |
| \( \text{underDomainOf p??} \) |
| \( \text{member p1198} \) |
| \( \text{isSubDomain p??} \) |
| \( \text{sayFunctionSelection p??} \) |
| \( \text{selectMms1 p??} \) |
| \( \text{sayFunctionSelectionResult p??} \) |
| \( \text{compiledLookup p1185} \) |
| \( \text{evalDomain p1075} \) |
| \( \text{wrapped2Quote p??} \) |
| \( \text{objVal p448} \) |
| \( \text{timedEvaluate p??} \) |
| \( \text{mkObjWrap p447} \) |
| \( \text{coerceFailure p??} \) |

---

defun coerceIntByMap

(defun |coerceIntByMap| (triple t2)
  (let (t1 top u1 u2 args mms fun code val)
    (declare (special |$coerceFailure| |$reportBottomUpFlag|))
    (setq t1 (|objMode| triple))
    (cond
      ((equal t2 t1) triple)
      (t
        (setq u2 (|deconstructT| t2)) ; compute t2 first because of Expression
        (cond
          ((eq 1 (|#| u2)) nil) ; no under domain
          (t
            (setq u1 (|deconstructT| t1))
            (cond
              ((eq 1 (|#| u1)) nil)
              ((nequal (caar u1) (caar u2)) nil); constructors not equal
              ((null (|valueArgsEqual?| t1 t2)) nil)
              (t ; handle a couple of special cases for subdomains of Integer
                (setq top (caar u1))
                (setq u1 (|underDomainOf| t1)))))
    ))
  )
)

---

---
(setq u2 (|underDomainOf| t2))
(cond
  ((and (|member| top
        '(|List| |Vector| |Segment| |Stream|
            |UniversalSegment| |Array|))
     (|isSubDomain| u1 u2))
    (mkObj (|objVal| triple) t2))
  (t
    (setq args (list (list '|Mapping| u2 u1) t1))
    (when |$reportBottomUpFlag|
      (|sayFunctionSelection| '|map| args t2 nil
        "coercion facility (map)")
    (setq mms (|selectMms1| '|map| t2 args args nil))
    (when |$reportBottomUpFlag|
      (|sayFunctionSelectionResult| '|map| args mms))
    (cond
      ((null mms) nil)
      (t
        (setq fun
          (|compiledLookup| '|map| (cdaar mms) (|evalDomain| (caaar mms))))
        (cond
          ((null fun) nil)
          (t
            (cond
              ((equal (car fun) #'|Undef|) nil)
              (t
                ; now compile a function to do the coercion
                (setq code
                  (list 'spadcall
                    (list 'cons
                      (list 'function '|coerceIntByMapInner|)
                      (mkq (cons u1 u2))
                      (|wrapped2Quote| (|objVal| triple))
                      (mkq fun))))
                ; and apply the function
                (setq val (catch '|coerceFailure| (|timedEvaluate| code)))
                (unless (equal val |$coerceFailure|)
                  (mkObjWrap val t2)))))))))))))))

defun coerceIntByMapInner

This is a helper function for coerceIntByMap which constructs a spadcall and then evaluates it. [coerceOrThrowFailure p695]

— defun coerceIntByMapInner —
(defun coerceIntByMapInner (arg g1)
  (|coerceOrThrowFailure| arg (car g1) (cdr g1)))

defun coerceOrThrowFailure

[coerceOrRetract p703]
[mkObjWrap p447]
[coercionFailure p695]
[objValUnwrap p448]

— defun coerceOrThrowFailure —

(defun coerceOrThrowFailure (value t1 t2)
  (let (result)
    (or (setq result (|coerceOrRetract| (mkObjWrap value t1) t2))
        (|coercionFailure|))
    (|objValUnwrap| result)))

—

defun coercionFailure

This does a throw on coercion failure. [coerceFailure p7?]

— defun coercionFailure —

(defun coercionFailure ()
  (declare (special $coerceFailure))
  (throw '|coerceFailure| $coerceFailure))

—

defun valueArgsEqual?

[u1,:u2] gets passed as the “environment”, which is why we have this slightly clumsy locution

This returns true if the object-valued arguments to t1 and t2 are the same under coercion
[getdatabase p1156]
[getConstructorSignature p7?]
[replaceSharps p1104]
defun valueArgsEqual? (t1 t2)
(let (coSig constrSig tl1 tl2 newVal done value trip)
  (setq coSig (cdr (getdatabase (car t1) 'cosig)))
  (setq constrSig (cdr (getConstructorSignature (car t1))))
  (setq tl1 (replaceSharps constrSig t1))
  (setq tl2 (replaceSharps constrSig t2))
  (cond
   ((null (member nil coSig)) t)
   (t
    (setq done nil)
    (setq value t)
    (loop for a1 in (cdr t1) for a2 in (cdr t2) for cs in coSig
       for m1 in tl1 for m2 in tl2 while (not done) do
      (cond
       ((null cs)
        (setq trip (mkObjWrap a1 m1))
        (setq newVal (coerceInt trip m2))
        (cond
         ((null newVal)
          (setq done t)
          (setq value nil))
         ((null (algEqual a2 (objValUnwrap newVal) m2))
          (setq done t)
          (setq value nil))))))
    value))))

defun algEqual
This function sees if 2 objects of the same domain are equal by using the = from the domain.
The objects should not be wrapped. [spadcall p?]

(spadcall object1 object2
  (|compiledLookupCheck| ' ' (list |$Boolean| ' ' $) (|evalDomain| domain))))

defun coerceIntFromUnion

— defun coerceIntFromUnion —

(defun coerceIntFromUnion (object t2)
  (|coerceInt| (|coerceUnion2Branch| object) t2))

defun coerceInt2Union

— defun coerceInt2Union —

(defun coerceInt2Union (object union)
  (let (unionDoms t1 val valp noCoerce)
    (declare (special |$String|))
    (setq unionDoms (|stripUnionTags| (cdr union)))
    (setq t1 (|objMode| object))
    (cond
      ((|member| t1 unionDoms) (|coerceBranch2Union| object union))
      (t
       (setq val (|objVal| object))
       (setq valp (|unwrap| val))
       (cond
        ((and (equal t1 |$String|) (|member| valp unionDoms))
         (|coerceBranch2Union| (mkObj val valp) union))
        (t
         (setq noCoerce t)
         (setq valp nil)
         (loop for d in unionDoms while noCoerce do
              (when (setq valp (|coerceInt| object d)) (setq noCoerce nil)))
         (when valp (|coerceBranch2Union| valp union))))))
defun coerceBranch2Union

[orderUnionEntries p]
[mkPredList p]
[stripUnionTags p]
[position p]
[keyedSystemError p]
[objMode p]
[objVal p]
[mkObjWrap p]
[removeQuote p]
[unwrap p]
[mkObj p]

| --- defun coerceBranch2Union --- |

(defun |coerceBranch2Union| (object union)
  (let ((predList doms p val tag)
    (setq doms (|orderUnionEntries| (cdr union)))
    (setq predList (|mkPredList| doms))
    (setq doms (|stripUnionTags| doms))
    (setq p (|position| (|objMode| object) doms))
    (cond
      ((equal p (- 1))
       (|keyedSystemError| "The type %1p is not branch of %2p"
         (list (|objMode| object) union)))
      (t
       (setq val (|objVal| object))
       (if (eq (car (setq tag (elt predlist p))) 'eqcar)
           (mkObjWrap (cons (|removeQuote| (third tag)) (|unwrap| val)) union)
           (mkObj val union))))))

---

defun coerceIntAlgebraicConstant

[|objMode p]
[|objValUnwrap p]
[|ofCategory p]
[mkObjWrap p]
[getConstantFromDomain p]

| --- defun coerceIntAlgebraicConstant --- |

(defun |coerceIntAlgebraicConstant| (object t2)
  (let (t1 val)
(setq t1 (|objMode| object))
(setq val (|objValUnwrap| object))
(cond
  ((and (|ofCategory| t1 '(|Monoid|))
        (|ofCategory| t2 '(|Monoid|))
        (equal val (|getConstantFromDomain| '(|One|) t1)))
   (mkObjWrap (|getConstantFromDomain| '(|One|) t2) t2))
  ((and (|ofCategory| t1 '(|AbelianMonoid|))
        (|ofCategory| t2 '(|AbelianMonoid|))
        (equal val (|getConstantFromDomain| '(|Zero|) t1)))
   (mkObjWrap (|getConstantFromDomain| '(|Zero|) t2) t2))))

---

**defun getConstantFromDomain**

The function `getConstantFromDomain` is used to look up the constants 0 and 1 from the given domainForm.

If `isPartialMode` returns true then the domain modemap contains the constant `EmptyMode` which indicates that the domain is not fully formed. In this case we return nil.

```
[isPartialMode p650]
[opOf p720]
[lassoc p720]
[getOperationAlistFromLisplib p769]
[getConstantFromDomain p699]
[throwKeyedMsg p720]
[spadcall p720]
[compiledLookupCheck p769]
[evalDomain p1075]

— defun getConstantFromDomain —
```

```
(defun |getConstantFromDomain| (form domainForm)
  (let (key entryList)
    (unless (|isPartialMode| domainForm)
      (setq key (|opOf| form))
      (setq entryList
        (lassoc key (|getOperationAlistFromLisplib| (car domainForm))))
      (cond
        ((null (eq (cdr entryList) nil))
         (cond
          ((eq key '|One|) (|getConstantFromDomain| (list '|1|) domainForm))
          ((eq key '|Zero|) (|getConstantFromDomain| (list '|0|) domainForm))
          (t
           (|throwKeyedMsg| "No such constant %1 in domain %2p ." form domainForm))))))
```
(t
   ; there should be exactly one item under this key of that form
   (spadcall
      ([compiledLookupCheck| key (caar entryList)
         ([evalDomain| domainForm]))))))

---

defun compareTypeLists

Rreturns true if every type in tl1 is equal or is a subdomain of the corresponding type in tl2

— defun compareTypeLists —

(defun |compareTypeLists| (tl1 tl2)
  (not
   (loop for t1 in tl1 for t2 in tl2
      do (when (null ([isEqualOrSubDomain| t1 t2]) (return t))))))

---

defun coerceIntX

Try to coerce a (List (None)) into a different domain [unwrap p??]
[underDomainOf p??]
[coerceInt p673]
[mkObjWrap p447]

— defun coerceIntX —

(defun |coerceIntX| (val t1 t2)
  (let (t0)
    (when (and (equal t1 '([List] [None])))
      (null ([unwrap| val]))
      (setq t0 ([underDomainOf| t2]))
      ([coerceInt| (mkObjWrap val (list '([List] t0)) t2)]))

---

defun coerceSubDomain

[getdatabase p1156]
[coerceSubDomain p700]
(defun coerceSubDomain (val tSuper tSub)
  (let (super)
    (unless (eq val ’$fromCoerceable$)
      (setq super (getdatabase (car tSub) ’superdomain))
      (cond
        ((equal (car super) tSuper)
         (coerceImmediateSubDomain val tSuper tSub (second super)))
        ((coerceSubDomain val tSuper (car super))
         (coerceImmediateSubDomain val (car super) tSub (second super)))))))

defun coerceImmediateSubDomain

(defun coerceImmediateSubDomain (val tSuper tSub pred)
  (when (funcall (getSubDomainPredicate tSuper tSub pred) val nil)
    (mkObj val tSub)))

defun getSubDomainPredicate

(defun getSubDomainPredicate (val tSuper tSub pred)
  (msubst tSuper tSub pred)
  (removeZeroOne tSuper tSub pred)
  (interpret tSuper tSub pred)
  (mkAtree tSuper tSub pred)
  (transferPropsToNode tSuper tSub pred)
  (selectLocalMms tSuper tSub pred)
  (lput tSuper tSub pred)
  ($env tSuper tSub pred)
  ($superHash tSuper tSub pred)
  ($Boolean tSuper tSub pred)
  ($InteractiveFrame tSuper tSub pred))
(defun getSubDomainPredicate (tSuper tSub pred)
  (let ((|$env| name decl arg predp defn op predfn)
    (declare (special |$env| |$superHash| |$Boolean| |$InteractiveFrame|))
    (setq |$env| |$InteractiveFrame|)
    (cond
      ((setq predfn (hget |$superHash| (cons tSuper tSub))) predfn)
      (t
        (setq name (gensym))
        (setq decl (list '|:| name (list '|Mapping| |$Boolean| tSuper)))
        (interpret decl nil)
        (setq arg (gensym))
        (setq predp (msubst arg '|#1| pred))
        (setq defn
          (list 'def (list name arg) '(nil nil) '(nil nil) (|removeZeroOne| predp)))
        (interpret defn nil)
        (setq op (|mkAtree| name))
        (transferPropsToNode name op)
        (setq predfn (cadar (|selectLocalMms| op name (list tSuper) |$Boolean|)))
        (hput |$superHash| (cons tSuper tSub) predfn))))

---

(defun absolutelyCanCoerceByCheating (t1 t2)
  (let (let1 let2)
    (declare (special |$Integer| |$SingleInteger|))
    (cond
      (((isEqualOrSubDomain t1 t2) t)
        (t
          (setq let1 (|deconstructT| t1))
          (setq let2 (|deconstructT| t2))
          (cond
            ((and (equal t1 |$SingleInteger|) (equal t2 |$Integer|)) t)
            ((or (atom t1) (atom t2)) nil)
            (t
              (setq let1 (|deconstructT| t1))
              (setq let2 (|deconstructT| t2))
              (cond
                ((and (equal (car let1) '(|Stream|))))
                )))
  )))
(equal (car let2) '((InfiniteTuple)))
(cond
  ((nequal (|#| (cdr let1)) (|#| (cdr let2))) nil)
  (t
      (every #'identity
          (loop for x1 in (cdr let1) for x2 in (cdr let2) collect
                ([absolutelyCanCoerceByCheating| x1 x2]))))
  ((nequal (car let1) (car let2)) nil)
  ((nequal (|#| (cdr let1)) (|#| (cdr let2))) nil)
  (t
      (every #'identity
          (loop for x1 in (cdr let1) for x2 in (cdr let2) collect
                ([absolutelyCanCoerceByCheating| x1 x2])))
      )))

---

defun coerceOrRetract

[coerceInteractive p672]
[retract p1203]

— defun coerceOrRetract —

(defun |coerceOrRetract| (z m)
  (prog (tp tt ans)
    (return
      (cond
        ((setq tp (|coerceInteractive| z m)) tp)
        (t
          (setq tt z)
          (setq ans nil)
          (do () (nil nil)
            (cond
              (ans (return ans))
              (t
                (setq tt (|retract| tt))
                (cond
                  ((eq tt '|failed|) (return ans))
                  (t (setq ans ([coerceInteractive| tt m]))))))
          ans)))))

---
defun retract2Specialization

Handle some specialization retraction cases, like matrices [objVal p448]
unwrap p??
[objMode p448]
mkObjWrap p447
goerceUnion2Branch p707
goerceInt p673
[remdup p??]
[varsInPoly p??]
mkObj p446
member p1198
retract p1203
[objValUnwrap p448]
[objMode p448]
[resolveTypeListAny p??]
isRectangularList p??
[get p??]
isPartialMode p650
$e p247
$QuotientField p645
$Symbol p645
$Integer p643
$Any p640
$NonNegativeInteger p644
$PositiveInteger p644

— defun retract2Specialization —

(defun |retract2Specialization| (object)
  (prog (val type dom obj dp bad vl tl ep vlp n D num den valp m)
    (declare (special $e $QuotientField $Symbol $Integer $Any $NonNegativeInteger $PositiveInteger))
    (return
      (seq
        (progn
          (setq val (|objVal| object))
          (setq valp (|unwrap| val))
          (setq type (|objMode| object))
          (cond
            ; type is Any
            (equal type $Any)
            (setq dom (car valp))
            (setq obj (cdr valp))
            (mkObjWrap obj dom))
            ; type is ['Union,:unionDoms]
            (eq (car type) '|Union|))
; type isnt ['List,dp]
((null (eq (car d) 'List))
 (cond
  ((equal d $PositiveInteger)
   (mkObj val (list 'List 'NonNegativeInteger)))
  ((equal d $NonNegativeInteger)
   (mkObj val (list 'List $Integer)))
  ((null valp) nil)
  (t
   (setq vl nil)
   (setq tl nil)
   (setq bad nil)
   (loop for e in valp while (not bad) do
     (cond
      ((equal (setq ep (retract (mkObjWrap e d))) 'failed)
       (setq bad t))
      (t
       (setq vl (cons (objValUnwrap ep) vl))
       (setq tl (cons (objMode ep) tl))))
   (cond
    (bad nil)
    ((equal (setq m (resolveTypeListAny tl)) d) nil)
    ((equal d m) nil)
    (t
     (setq vlp nil)
     (setq ep t)
     (loop for e in vl for tt in tl while ep do
       (cond
        ((equal tt m) (setq vlp (cons e vlp)))
        (t
         (setq ep (coerceInt (mkObjWrap e tt) m))
         (when ep (setq vlp (cons (objValUnwrap ep) vlp)))))
     (mkObjWrap vlp (list 'List m))))
  )
  ((equal dp $PositiveInteger)
   (mkObj val (list 'List (list 'List 'NonNegativeInteger)))
  )
  ((equal dp $NonNegativeInteger)
   (mkObj val (list 'List (list 'List $Integer)))
  )
  ((or (eq (car dp) 'Variable)
    (eq (car dp) 'OrderedVariableList))
   (coerceInt object (list 'List (list 'List $Symbol)))
  )
  (t
   (setq n (|#| valp))
   (setq m (|#| (elt valp 0)))
   (cond
    ((null (listRectangularList valp n m)) nil)
    (t (coerceInt object (list 'Matrix dp)))))))
; type is ['Expression,d]
((eq (car type) 'Expression)
 (setq num (car valp))
 (setq den (cdr valp)))
(cond
  ((null (equal (car num) 0)) nil)
  ((null (equal (car den) 0)) nil)
  (t
    (mkObjWrap (cons (cdr num) (cdr den))
      (list (quotientField) (second type)))))))

; type is ['SimpleAlgebraicExtension,k,rep,..]
; try to retract as an element of rep and see if we can get an element of k
((eq (car type) '|SimpleAlgebraicExtension|)
  (setq valp (retract (mkObj val (third type))))
  (do ()
    ((null (and (nequal valp '|failed|)
                  (nequal (objMode valp) (second type))))
      nil)
    (setq valp (retract valp)))
  (unless (equal valp '|failed|) valp)))

; type is ['UnivariatePuiseuxSeries,coef,var,cen]
((eq (car type) '|UnivariatePuiseuxSeries|)
  (coerceInt object
    (list '|UnivariateLaurentSeries| (second type) (third type) (fourth type))))

; type is ['UnivariateLaurentSeries,coef,var,cen]
((eq (car type) '|UnivariateLaurentSeries|)
  (coerceInt object
    (list '|UnivariateTaylorSeries| (second type) (third type) (fourth type))))

; type is ['FunctionCalled,name]
((eq (car type) '|FunctionCalled|)
  (cond
    ((null (setq m (get (second type) '|mode| |$e|))) nil)
    ((isPartialMode m) nil)
    (t (mkObj val m)))
    (t nil)))))))

defun coerceUnion2Branch

[orderUnionEntries p??]
[objMode p448]
[mkPredList p??]
[stripUnionTags p708]
[objValUnwrap p448]
[evalSharpOne p708]
[mkObj p446]
[objVal p448]
--- defun coerceUnion2Branch ---

(defun coerceUnion2Branch (object)
  (let (predList doms valp predicate targetType)
    (setq doms (orderUnionEntries (cdr (objMode object))))
    (setq predList (mkPredList doms))
    (setq doms (stripUnionTags doms))
    (setq valp (objValUnwrap object))
    (loop for typ in doms for pred in predList while (not targetType) do
      (when (evalSharpOne pred valp)
        (setq predicate pred)
        (setq targetType typ)))
    (cond
      ((null targetType)
        (keyedSystemError "Cannot determine branch of Union." nil))
      ((eq (car predicate) 'eqcar) (mkObjWrap (cdr valp) targetType))
      (t (mkObj (objVal object) targetType))))

---

defun stripUnionTags

--- defun stripUnionTags ---

(defun stripUnionTags (doms)
  (loop for dom in doms
    collect (if (eq (first dom) ':|) (third dom) dom)))

---

defun evalSharpOne

--- defun evalSharpOne 0 ---

(defun evalSharpOne (x #1)
  (declare (special #1))
  (eval `(let() (declare (special #1))) ,x))

---
defun retractUnderDomain

[underDomainOf p??]
[deconstructT p??]
[nequal p??]
[constructT p??]
[coerceInt p673]

— defun retractUnderDomain —

(defun retractUnderDomain (object type underDomain)
  (let (ud let1 typep objectp)
    (cond
      ((null (setq ud (underDomainOf underDomain))) 'failed)
      (t
       (setq let1 (deconstructT type))
       (cond
         ((nequal 1 (cdr let1))) 'failed)
         ((nequal 1 (car let1))) 'failed)
      (t
       (setq typep (constructT (car let1) (list ud)))
       (cond
         ((setq objectp (coerceInt object typep)) objectp)
         (t 'failed)))))))

—

defun coerceRetract

[objValUnwrap p448]
[objMode p448]
[isEqualOrSubDomain p668]
[mkObjWrap p447]
[retractByFunction p710]
[getl p1200]
[canFuncall? p1197]
[$coerceFailure p??]
[$SingleInteger p647]
[$OutputForm p642]
[$Symbol p645]
[$Integer p643]
[coerceFailure p??]

— defun coerceRetract —

(defun coerceRetract (object t2)
(let (val t1 fun c)
  (declare (special |$coerceFailure| |$OutputForm| |$Symbol| |$Integer|
             |$SingleInteger|))
  (cond
    ((eq (setq val (|objValUnwrap| object)) '|$fromCoerceable$|) nil)
    (t
     (setq t1 (|objMode| object))
     (cond
      ((equal t2 |$OutputForm|) nil)
      ((and (|isEqualOrSubDomain| t1 |$Integer|)
            (equal t2 |$SingleInteger|)
            (typep val 'fixnum))
       (mkObjWrap val t2))
      ((equal t1 |$Integer|) nil)
      ((equal t1 |$Symbol|) nil)
      ((equal t1 |$OutputForm|) nil)
      ((setq c (|retractByFunction| object t2)) c)
      ((consp t1)
       (setq fun
         (or (getl (car t1) '|retract|)
             (intern (strconc "retract" (princ-to-string (car t1)))))))
      (when (canFuncall? fun)
       (put (car t1) '|retract| fun)
       (setq c (catch '|coerceFailure| (funcall fun object t2)))
       (unless (equal c |$coerceFailure|) c)))))))

(defun retractByFunction
  [objValUnwrap p448]
  [sayFunctionSelection p??]
  [findFunctionInDomain p??]
  [orderMms p??]
  [sayFunctionSelectionResult p??]
  [evalDomain p1075]
  [compiledLookup p1185]
  [coerceUnion2Branch p707]
  [mkObjWrap p447]
  [spadcall p??]
  [objMode p448]
  [$reportBottomUpFlag p941]
  [$dollar p??]

  —— defun retractByFunction ——

(defun |retractByFunction| (object u)
(let (($reportBottomUpFlag| $ tt val target funName mms dcVector fun objectp)
  (declare (special |$reportBottomUpFlag| $))
  (setq tt (|objMode| object))
  (setq val (|objValUnwrap| object))
  (setq target (list '|Union| u "failed"))
  (setq funName '|retractIfCan|)
  (when |$reportBottomUpFlag|
    (|sayFunctionSelection| funName (list tt) target
      nil "coercion facility (retraction)"))
  (when
    (setq mms
      (append
        (|findFunctionInDomain| funName tt target (list tt) (list tt) nil t)
        (|findFunctionInDomain| funName u target (list tt) (list tt) nil t)))
    (setq mms (|orderMms| funName mms (list tt) (list tt) target)))
  (when |$reportBottomUpFlag|
    (|sayFunctionSelectionResult| funName (list tt) mms))
  (when mms
    (setq dcVector (|evalDomain| (caaar mms)))
    (setq fun (|compiledLookup| funName (list target tt) dcVector))
    (cond
      ((null fun) nil)
      ((equal (car fun) #'|Undef|) nil)
      (t
       (setq $ dcVector)
       (setq objectp
         (|coerceUnion2Branch| (mkObjWrap (spadcall val fun) target))
       (when (equal u (|objMode| objectp)) objectp))))))}
Chapter 25

System Command Handling

The system commands are the top-level commands available in Axiom that can all be invoked by prefixing the symbol with a closed-paren. Thus, to see the copyright you type:

)`copyright

New commands need to be added to this table. The command invoked will be the first entry of the pair and the “user level” of the command will be the second entry.

See:

- The “abbreviations” (25.3 p 754) command
- The “boot” (25.4 p 758) command
- The “browse” (25.5 p 759) command
- The “cd” (25.11 p 764) command
- The “clear” (25.12 p 767) command
- The “close” (25.13 p 777) command
- The “compile” (25.14 p 779) command
- The “copyright” (25.15 p 787) command
- The “credits” (25.16 p 788) command
- The “display” (25.18 p 796) command
- The “edit” (25.19 p 804) command
- The “fin” (25.20 p 808) command
• The “frame” (2.5 p 10) command
• The “help” (25.21 p 812) command
• The “history” (25.23 p 821) command
• The “lisp” (25.27 p 867) command
• The “library” (34.1 p 1159) command
• The “license” (25.26 p 866) command
• The “load” (?? p ??) command
• The “itrace” (25.28 p 868) command
• The “pquit” (25.29 p 870) command
• The “quit” (25.30 p 872) command
• The “read” (25.31 p 875) command
• The “regress” (25.32 p 882) command
• The “savesystem” (25.33 p 891) command
• The “set” (25.51 p 1013) command
• The “show” (25.52 p 1019) command
• The “spool” (25.53 p 1033) command
• The “summary” (25.54 p 1035) command
• The “synonym” (25.55 p 1037) command
• The “system” (25.56 p 1041) command
• The “tangle” (25.57 p 1042) command
• The “trace” (25.59 p 1045) command
• The “trademark” (25.58 p 1044) command
• The “undo” (25.60 p 1072) command
• The “what” (25.62 p 1091) command
• The “with” (?? p ??) command
• The “workfiles” (25.63 p 1099) command
25.1 Variables Used

defvar $systemCommands

— initvars —

(defvar $systemCommands nil)

— postvars —

(eval-when (eval load)
 (setq $systemCommands
       '(abbreviations . compiler)
       (boot . development)
       (browse . development)
       (cd . interpreter)
       (clear . interpreter)
       (close . interpreter)
       (compiler . compiler)
       (copyright . interpreter)
       (credits . interpreter)
       (describe . interpreter)
       (display . interpreter)
       (edit . interpreter)
       (fin . development)
       (frame . interpreter)
       (help . interpreter)
       (history . interpreter)
       (lisp . development)
       (library . interpreter)
       (license . interpreter)
       (load . interpreter)
       (ltrace . interpreter)
       (pquit . interpreter)
       (quit . interpreter)
       (read . interpreter)
       (regress . interpreter)
       (savesystem . interpreter)
       (set . interpreter)
       (show . interpreter)
       (spool . interpreter)
       (summary . interpreter)
       (synonym . interpreter)
defvar $syscommands

This table is used to look up a symbol to see if it might be a command.

----

— initvars —

(defvar $syscommands nil)

----

— postvars —

(eval-when (eval load)
  (setq $syscommands (mapcar #'car |$systemCommands|)))

----

defvar $noParseCommands

This is a list of the commands which have their arguments passed verbatim. Certain func-
tions, such as the lisp function need to be able to handle all kinds of input that will not be
acceptable to the interpreter.

----

— initvars —

(defvar |$noParseCommands| nil)

----

— postvars —
25.2 Functions

defun handleNoParseCommands

The system commands given by the global variable $noParseCommands$ require essentially no preprocessing/parsing of their arguments. Here we dispatch the functions which implement these commands.

There are four standard commands which receive arguments

- boot
- lisp
- synonym
- system

There are six standard commands which do not receive arguments –

- quit
- fin
- pquit
- credits
- copyright
- trademark

As these commands do not necessarily exhaust those mentioned in $noParseCommands$, we provide a generic dispatch based on two conventions: commands which do not require an argument name themselves, those which do have their names prefixed by “np”. This makes it possible to dynamically define new system commands provided you handle the argument parsing.
defun Handle a top level command

(defun doSystemCommand (string)
  (let (line tok unab optionList)
    (declare (special line \$tokenCommands\ \$noParseCommands\))
    (setq string (concat "\)" (expand-tabs string)))
    (setq line string)
    (processSynonyms)
    (setq string line)
    (setq string (substring string 1 nil))
    (cond
      ((string= string "") nil)
      (t
       (setq tok (getFirstWord string))
       (cond
        (tok
         (setq unab (unAbbreviateKeyword tok))
         (cond
          ((member unab \$noParseCommands\)
           (handleNoParseCommands unab string))
          (t
           (setq optionList (splitIntoOptionBlocks string))
           (cond
            ((member unab \$tokenCommands\)
             (handleTokensizeSystemCommands unab optionList))
            (t
             (handleParsedSystemCommands unab optionList) nil)))))))
    (t nil))))

---

defun doSystemCommand
25.2. FUNCTIONS

defun Split block into option block

(stripSpaces p743)

— defun splitIntoOptionBlocks —

(defun splitIntoOptionBlocks (str)
   (let ((inString nil) (block nil) (blockStart 0) (parenCount 0) blockList)
      (dotimes (i (1- (length str)))
         (cond
            ((char= (elt str i) "\n") (setq inString nil))
            (t
             (when (and (char= (elt str i) "\( ") (null inString)
                 (incf parenCount))
               (when (and (char= (elt str i) ")") (null inString))
                 (decf parenCount)))
            (when (and (char= (elt str i) ")") (null inString)
                 (= parenCount -1))
               (setq block (stripSpaces (subseq str blockStart i)))
               (setq blockList (cons block blockList))
               (setq blockStart (1+ i))
               (setq parenCount 0))))
      (setq blockList (cons (stripSpaces (subseq str blockStart)) blockList))
      (nreverse blockList)))

defun Tokenize a system command

[dumbTokenize p739]
[tokTran p739]
[systemCommand p720]

— defun handleTokenizeSystemCommands —

(defun handleTokenizeSystemCommands (unabr optionList)
   (declare (ignore unabr))
   (let (parcmd)
      (setq optionList (mapcar #'(lambda (x) (dumbTokenize x)) optionList))
      (setq parcmd
         (mapcar #'(lambda (opt) (mapcar #'(lambda (tok) (tokTran tok)) opt))
            optionList))
      (when parcmd (systemCommand parcmd))))
defun Handle system commands

You can type ")?" and see trivial help information. You can type ")? compile" and see compiler related information

[selectOptionLC p751]
[helpSpad2Cmd p813]
[selectOption p751]
[commandsForUserLevel p720]
[options p??]
[e p247]
[systemCommands p715]
[syscommands p716]
[CategoryFrame p??]

— defun systemCommand —

(defun systemCommand (cmd)
  (let ((options e op argl options fun)
    (declare (special options e systemCommands syscommands CategoryFrame))
    (setq op (caar cmd))
    (setq argl (cdar cmd))
    (setq options (cdr cmd))
    (setq e options)
    (setq CategoryFrame)
    (setq fun (selectOptionLC op systemCommands 'commandError))))
  (if (and argl (eq (elt argl 0) ?) (not (eq fun synonym)))
    (helpSpad2Cmd (cons fun nil))
    (progn
      (setq fun
        (selectOption fun (commandsForUserLevel systemCommands)
        'commandUserLevelError)))
    (funcall fun argl)))))

— defun Select commands matching this user level —

defun Select commands matching this user level

The $UserLevel contains one of three values: compiler, development, or interpreter. This variable is used to select a subset of commands from the list stored in $systemCommands, representing all of the commands that are valid for this level. [satisfiesUserLevel p723]

— defun commandsForUserLevel —

(defun commandsForUserLevel (arg)
  (let (c)
(dolist (pair arg)
  (when (|satisfiesUserLevel| (cdr pair))
    (setq c (cons (car pair) c)))
  (nreverse c)))

---

defun No command begins with this string
[commandErrorMessage p721]

— defun commandError —

(defun |commandError| (x u)
  (|commandErrorMessage| '|command| x u))

---

defun No option begins with this string
[commandErrorMessage p721]

— defun optionError —

(defun |optionError| (x u)
  (|commandErrorMessage| '|option| x u))

---

defvar $oldline

— initvars —

(defvar $oldline nil "used to output command lines")

---

defun No command/option begins with this string
[commandAmbiguityError p724]
[sayKeyedMsg p27]
---

**defun commandErrorMessage**

```lisp
(defun commandErrorMessage (kind x u)
  (declare (special $oldline line))
  (setq $oldline line)
  (if u
      (commandAmbiguityError kind x u)
    (progn
      (sayKeyedMsg "No %1 begins with %2 ." (list kind x))
      (terminateSystemCommand))))
```

---

**defun Option not available at this user level**

**defun optionUserLevelError**

```lisp
(defun optionUserLevelError (x u)
  (userLevelErrorMessage 'option x u))
```

---

**defun Command not available at this user level**

**defun commandUserLevelError**

```lisp
(defun commandUserLevelError (x u)
  (userLevelErrorMessage 'command x u))
```

---

**defun Command not available error message**

**commandAmbiguityError**

```lisp
(defun commandAmbiguityError (kind x u)
  (sayKeyedMsg "No %1 begins with %2 ." (list kind x))
  (terminateSystemCommand)))
```
25.2. FUNCTIONS

— defun userLevelErrorMessage —

(defun userLevelErrorMessage (kind x u)
  (declare (special $UserLevel))
  (if u
      (commandAmbiguityError kind x u)
      (progn
       (sayKeyedMsg
        "Your %1 is ambiguous. The following are abbreviated by %2 :
        (list $UserLevel kind))
       (terminateSystemCommand))))

— defun satisfiesUserLevel 0 —

(defun satisfiesUserLevel (x)
  (declare (special $UserLevel))
  (cond
   ((eq x 'interpreter) t)
   ((eq $UserLevel 'interpreter) nil)
   ((eq x 'compiler) t)
   ((eq $UserLevel 'compiler) nil)
   (t t))

— defun hasOption —

(defun hasOption (al opt)
  (let ((optPname (pname opt)) found)
    (loop for pair in al do
      (when (stringPrefix? (pname (car pair)) optPname) (setq found pair))
      (when found (return)))
    (t nil))
CHAPTER 25. SYSTEM COMMAND HANDLING

---

defun terminateSystemCommand

- defun terminateSystemCommand —

(defun |terminateSystemCommand| nil (tersyscommand))

---

defun Terminate a system command

- defun tersyscommand —

(defun tersyscommand ()
  (let (chr tok)
    (fresh-line)
    (setq chr 'endoflinechr)
    (setq tok 'end_unit)
    (|spadThrow|)))

---

defun commandAmbiguityError

- defun commandAmbiguityError —

(defun |commandAmbiguityError| (kind x u)
  (|sayKeyedMsg| (kind x u)
   (|sayKeyedMsg|)
   "Your %1 is ambiguous. The following are abbreviated by %2 :"

)
25.2. FUNCTIONS

(defun getParserMacroNames)

The $pfMacros is a list of all of the user-defined macros.

(defun getParserMacroNames ()
  (declare (special $pfMacros))
  (remove-duplicates (mapcar #'car $pfMacros))))

— defun getParserMacroNames 0 —

(defun clearParserMacro)

Note that if a macro is defined twice this will clear the last instance. Thus:

a ==> 3
a ==> 4
)d macros
a ==> 4
)clear prop a
)d macros
a ==> 3
)clear prop a
)d macros
nil

(ifcdr p??)
[assoc p??]
[remalist p??]
[$pfMacros p324]

— defun clearParserMacro —

(defun clearParserMacro (macro)
  (declare (special $pfMacros))
  (when (ifcdr (|assoc| macro $pfMacros))))
defun displayMacro
(isInterpMacro p) [sayBrightly p]
(bright p)
(strconc p)
(object2String p)
(mathprint p)
($op p)

— defun displayMacro —

(defun |displayMacro| (name)
(let (|$op| m body args)
(declare (special |$op|))
(setq m (|isInterpMacro| name))
(cond
((null m)
 (|sayBrightly|
 (cons " " (append (|bright| name)
 (cons "is not an interpreter macro." nil))))))
(t
(setq |$op| (strconc "macro " (|object2String| name)))
(setq args (car m))
(setq body (cdr m))
(setq args
 (cond
 ((null args) nil)
 ((null (cdr args)) (car args))
 (t (cons ’|Tuple| args))))
 (|mathprint| (cons ’map (cons (|cons| args body) nil))))))

— defun displayWorkspaceNames —

defun displayWorkspaceNames
(getInterpMacroNames p) [getParserMacroNames p]
[sayMessage p]
[msort p]
25.2. FUNCTIONS

---

(defun displayWorkspaceNames ()
  (let (pmacs names imacs)
    (setq imacs (getInterpMacroNames))
    (setq pmacs (getParserMacroNames))
    (let ((msg ("Names of User-Defined Objects in the Workspace:"))
      (names (msort (append (getWorkspaceNames) pmacs)))
      (if names
        (sayAsManyPerLineAsPossible (mapcar #'object2String names))
      (sayBrightly " * None *"))
      (setq imacs (setdifference imacs pmacs))
      (when imacs
        (sayMessage "Names of System-Defined Objects in the Workspace:")
        (sayAsManyPerLineAsPossible (mapcar #'object2String imacs))))))

---

defun getWorkspaceNames

(defun getWorkspaceNames ()
  (declare (special $InteractiveFrame))
  (nmsort (loop for g2 in (caar $InteractiveFrame) collect (car g2))))

---

defun fixObjectForPrinting

The $msgdbPrims variable is set to:

(%b %d %l %i %u %U %n %x %c %rj)

--- defun fixObjectForPrinting ---

(defun |fixObjectForPrinting| (v)
  (let (vp)
    (declare (special |$msgdbPrims|))
    (setq vp (|object2Identifier| v))
    (cond
      ((eq vp '%) "\%")
      ((|member| vp |$msgdbPrims|) (strconc "\" (pname vp)))
      (t v))))

defun displayProperties,sayFunctionDeps
 ;displayProperties(option,l) ==
 $dependentAlist : local := nil
 $dependeeAlist : local := nil
 [opt,:vl] := (l or ['properties])
 imacs := getInterpMacroNames()
 pmacs := getParserMacroNames()
 macros := REMDUP append(imacs, pmacs)
 if vl is ['all] or null vl then
  vl := MSORT append(getWorkspaceNames(),macros)
 if $frameMessages then sayKeyedMsg("S2IZ0065",[$interpreterFrameName])
 null vl =>
 null $frameMessages => sayKeyedMsg("S2IZ0066",NIL)
 sayKeyedMsg("S2IZ0067",[$interpreterFrameName])
 interpFunctionDepAlists()
 for v in vl repeat
  isInternalMapName(v) => 'iterate
  pl := getIProplist(v)
  option = 'flags => getAndSay(v,"flags")
  option = 'value => displayValue(v,getI(v,'value),nil)
  option = 'condition => displayCondition(v,getI(v,"condition"),nil)
  option = 'mode => displayMode(v,getI(v,'mode),nil)
  option = 'type => displayType(v,getI(v,'value),nil)
  option = 'properties =>
   v = "--flags--" => nil
   pl is [ ['cacheInfo,..],.. ] => nil
   vl := fixObjectForPrinting(v)
25.2. FUNCTIONS

null pl =>
  v in pmacs =>
    sayMSG '" This is a user-defined macro."
    displayParserMacro v
  isInterpMacro v =>
    sayMSG '" This is a system-defined macro."
    displayMacro v
  sayMSG '" none"
  propsSeen:: nil
for [prop,:val] in pl | MEMQ(prop,propsSeen) and val repeat
  prop in '(alias generatedCode IS_-GENSYM mapBody localVars) =>
    nil
  prop = 'condition =>
    displayCondition(prop,val,true)
  prop = 'recursive =>
    sayMSG '" This is recursive."
  prop = 'isInterpreterFunction =>
    sayMSG '" This is an interpreter function."
    sayFunctionDeps v where
    sayFunctionDeps x ==
      if dependents := GETALIST($dependentAlist,x) then
        null rest dependents =>
          sayMSG '" The following function or rule ",
            '"depends on this:",.bright first dependents]
          sayMSG '" The following functions or rules depend on this:";
          msg := ["%b",""
            for y in dependents repeat msg := [" ",y,:msg]
          sayMSG [::nreverse msg,"%d"]
        if dependees := GETALIST($dependeeAlist,x) then
          null rest dependees =>
            sayMSG '" This depends on the following function ",
              '"or rule:",.bright first dependees]
            sayMSG '" This depends on the following functions or rules:";
            msg := ["%b",""
              for y in dependees repeat msg := [" ",y,:msg]
            sayMSG ::nreverse msg,"%d"]
          prop = 'isInterpreterRule =>
            sayMSG '" This is an interpreter rule."
            sayFunctionDeps v
          prop = 'localModemap =>
            displayModemap(v,val,true)
          prop = 'mode =>
            displayMode(prop,val,true)
          prop = 'value =>
            val => displayValue(v,val,true)
          sayMSG [" ",prop,: ",val]
        propsSeen:: [prop,:propsSeen]
CHAPTER 25. SYSTEM COMMAND HANDLING

; sayKeyedMsg("S2IZ0068",[option])
; terminateSystemCommand()

[seq p??]
[getalist p??]
[exit p??]
[sayMSG p29]
[bright p??]
[$dependeeAlist p??]
[$dependentAlist p??]

— defun displayProperties,sayFunctionDeps —

(defun |displayProperties|,|sayFunctionDeps| (x)
  (prog (dependents dependees msg)
    (declare (special |$dependeeAlist| |$dependentAlist|))
    (return
      (seq
        (if (setq dependents (getalist |$dependentAlist| x))
          (seq
            (if (null (cdr dependents))
              (exit
                (|sayMSG| (cons " The following function or rule "
                  (cons "depends on this:" (|bright| (car dependents)))))))
              (|sayMSG| " The following functions or rules depend on this:"))
           (setq msg (cons " " nil))
           (do ((G166397 dependents (cdr G166397)) (y nil))
             ((or (atom G166397) (progn (setq y (car G166397)) nil)) nil)
             (seq (exit (setq msg (cons " " (cons y msg)))))
           (exit (|sayMSG| (append (nreverse msg) (cons '|%d| nil))))
          nil)
          (exit
            (if (setq dependees (getalist |$dependeeAlist| x))
              (seq
                (if (null (cdr dependees))
                  (exit
                    (|sayMSG| (cons " This depends on the following function "
                      (cons "or rule:" (|bright| (car dependees)))))))
                  (|sayMSG| " This depends on the following functions or rules:"))
               (setq msg (cons " " nil))
               (do ((G166406 dependees (cdr G166406)) (y nil))
                 ((or (atom G166406) (progn (setq y (car G166406)) nil)) nil)
                 (seq (exit (setq msg (cons " " (cons y msg))))))
               (exit (|sayMSG| (append (nreverse msg) (cons '|%d| nil))))
            nil)))))))

| |


25.2. FUNCTIONS

```lisp
defun displayValue |
   (defun |displayValue| (|$op| u omitVariableNameIfTrue)
   (declare (special |$op|))
   (let (expr op rhs label labmode)
        (declare (special |$EmptyMode|))
        (if (null u)
            (|sayMSG|
             (list '| Value of | (|fixObjectForPrinting| (pname |$op|)) ": (none)"))
            (progn
                (setq expr (|objValUnwrap| u))
                (if (or (and (consp expr) (progn (setq op (qcar expr)) t) (eq op 'map))
                          (equal (|objMode| u) |$EmptyMode|))
                   (|displayRule| |$op| expr)
                   (progn
                        (cond
                            (omitVariableNameIfTrue
                             (setq rhs ": ")
                             (setq label "Value (has type "))
                            t
                            (setq rhs ": ")
                            (setq label (strconc "Value of " (pname |$op|) ": ")))
                        (setq labmode (|prefix2String| (|objMode| u))
                         (when (atom labmode) (setq labmode (list labmode)))
                         (if (eq (getdatabase expr 'constructorKind) '|domain|)
                             (|sayMSG| (|concat| " " label labmode rhs (|form2String| expr))
                             (|mathprint|
                             (cons 'concat
```
(cons label
 (append lbumode
 (cons rhs
  (cons (outputFormat expr (objMode u)) nil)))))))

---

defun displayType

[sayMSG p29]
[fixObjectForPrinting p727]
[pname p1195]
[prefix2String p??]
[objMode p448]
[concat p1197]
[$op p??]

— defun displayType —

(defun |displayType| (|$op| u omitVariableNameIfTrue)
 (declare (special |$op|) (ignore omitVariableNameIfTrue))
 (let (type)
   (if (null u)
     (|sayMSG|
      (list " Type of value of " (|fixObjectForPrinting| (pname |$op|))
        ": (none)"))
    (progn
     (setq type (|prefix2String| (|objMode| u)))
     (when (atom type) (setq type (list type)))
     (|sayMSG|
      (concat
       (cons " Type of value of "
        (cons (|fixObjectForPrinting| (pname |$op|))
          (cons ": " type)))))
     nil)))))

---

defun getAndSay

[getI p??]
[sayMSG p29]

— defun getAndSay —
25.2. FUNCTIONS

(defun getAndSay (v prop)
  (let (val)
    (if (setq val (getI v prop))
      (sayMSG (cons '1 val) (cons '2 nil))
      (sayMSG (cons '1 none) (cons '2 nil))))

(defun displayProperties
getInterpMacroNames p
getParserMacroNames p
remdup p
qcdr p
qcar p
msort p
getWorkspaceNames p
sayKeyedMsg p
interpFunctionDepAlists p
isInternalMapName p
getIProplist p
getAndSay p
displayValue p
getI p
displayCondition p
displayMode p
displayType p
fixObjectForPrinting p
sayMSG p
bright p
prefix2String p
member p
displayParserMacro p
isInterpMacro p
displayMacro p
displayProperties,sayFunctionDeps p
displayModemap p
exit p
seq p
terminateSystemCommand p
$dependentAlist p
$dependeeAlist p
$frameMessages p
$interpreterFrameName p
— defun displayProperties —

(defun displayProperties (option al)
  (let ((|$dependentAlist| |$dependeeAlist| tmp1 opt imacs pmacs macros vl pl
         tmp2 vone prop val propsSeen)
    (declare (special |$dependentAlist| |$dependeeAlist| |$frameMessages|
               |$interpreterFrameName|)))
  (setq |$dependentAlist| nil)
  (setq |$dependeeAlist| nil)
  (setq tmp1 (or al (cons '|properties| nil)))
  (setq opt (car tmp1))
  (setq vl (cdr tmp1))
  (setq imacs (|getInterpMacroNames|))
  (setq pmacs (|getParserMacroNames|))
  (setq macros (remdup (append imacs pmacs)))
  (when (or
      (and (consp vl) (eq (qcdr vl) nil) (eq (qcar vl) '|all|))
      (null vl))
      (setq vl (msort (append (|getWorkspaceNames|) macros))))
  (when |$frameMessages|
      (|sayKeyedMsg| "The name of the current frame is %1 ."
        (cons |$interpreterFrameName| nil)))
  (cond
    ((null vl)
      (if (null |$frameMessages|)
        (|sayKeyedMsg| "The workspace is empty." nil))
        (|sayKeyedMsg| "The current frame, %1 , is empty."
          (cons |$interpreterFrameName| nil))))
  (t
    (interpFunctionDepAlists))
  (do ((G166440 vl (cdr G166440)) (v nil))
      ((or (atom G166440) (progn (setq v (car G166440)) nil)) nil)
    (seq (exit
      (cond
        ((isInternalMapName| v) '|iterate|)
        (t
          (setq pl (|getIProplist| v))
          (cond
            ((eq option '|flags|)
              (getAndSay| v 'flags|))
            ((eq option '|value|)
              (displayValue| v (getI| v '|value|) nil))
            ((eq option '|condition|)
              (displayCondition| v (getI| v '|condition|) nil))
            ((eq option '|mode|)
              (displayMode| v (getI| v '|mode|) nil))
            ((eq option '|type|)
              (displayType| v (getI| v '|value|) nil))
            ((eq option '|properties|)
              (cond

    )
    )
  )
)
((eq v '|--flags--|)
 nil)
((and (consp pl)
 (progn
 (setq tmp2 (qcar pl))
 (and (consp tmp2) (eq (qcar tmp2) '|cacheInfo|)))
 nil)
(t
 (setq vone (|fixObjectForPrinting| v))
(|sayMSG|
 (cons "Properties of"
 (append (|bright| (|prefix2String| vone)) (cons ":" nil))))
(cond
 ((null pl)
 (cond
 (((|member| v pmacs)
 (|sayMSG| " This is a user-defined macro.")
 (|displayParserMacro| v))
 ((|isInterpMacro| v)
 (|sayMSG| " This is a system-defined macro.")
 (|displayMacro| v))
 (t
 (|sayMSG| " none")))))
(t
 (setq propsSeen nil)
 (do ((G166451 pl (cdr G166451)) (G166425 nil))
 (or (atom G166451)
 (progn (setq G166425 (car G166451)) nil)
 (progn
 (setq prop (car G166425))
 (setq val (cdr G166425))
 (G166425)
 nil))))
(nil)
(seq
 (exit
 (cond
 ((and (null (member prop propsSeen)) val)
 (cond
 (((|member| prop
 '([alias] |generatedCode| IS-GENSYM
 |mapBody| |localVars|)))
 nil)
 ((eq prop '|condition|)
 (|displayCondition| prop val t))
 ((eq prop '|recursive|)
 (|sayMSG| " This is recursive.")
 (|displayProperties,sayFunctionDeps| v)))
))})
CHAPTER 25. SYSTEM COMMAND HANDLING

(defun displayParserMacro
  [pfPrintSrcLines p??]
  [$pfMacros p324]
  — defun displayParserMacro —

  (defun |displayParserMacro| (m)
    (let ((m (assq m |$pfMacros|)))
      (declare (special |$pfMacros|))
      (when m (|pfPrintSrcLines| (caddr m)))))

———

(defun displayCondition
  [bright p??]
  [sayBrightly p??]
  [concat p1197]
  [pred2English p??]
  — defun displayCondition —

  (defun |displayCondition| (v condition giveVariableIfNil)
(let (varPart condPart)
  (when giveVariableIfNil (setq varPart (cons '|of| (|bright| v))))
  (setq condPart (or condition '|true|))
  (|sayBrightly|
   (|concat| 'condition varPart '|: |(|pred2English| condPart))))

---

defun interpFunctionDepAlists

[putalist p??]
[getalist p??]
[getFlag p??]
[$e p247]
[$dependeeAlist p??]
[$dependentAlist p??]
[$InteractiveFrame p23]

— defun interpFunctionDepAlists —

(defun |interpFunctionDepAlists| ()
  (let ((|$e|))
    (declare (special |$e| |$dependeeAlist| |$dependentAlist| |$InteractiveFrame|))
    (setq |$e| |$InteractiveFrame|)
    (setq |$dependentAlist| (cons (cons nil nil) nil))
    (setq |$dependeeAlist| (cons (cons nil nil) nil))
    (mapcar #'(lambda (dep)
      (let (dependee dependent)
        (setq dependent (first dep))
        (setq dependent (second dep))
        (setq |$dependentAlist|)
          (putalist |$dependentAlist| dependee
            (cons dependent (getalist |$dependentAlist| dependee))))
        (setq |$dependeeAlist|)
          (putalist |$dependeeAlist| dependent
            (cons dependee (getalist |$dependeeAlist| dependent))))
      (|getFlag| '|$dependencies|)))

——
defun displayModemap

[bright p??]
sayBrightly p??
concat p1197
(formatSignature p??]

—— defun displayModemap ——

(defun displayModemap (v val giveVariableIfNil)
(labels (g (v mm giveVariableIfNil)
  (let (local signature fn varPart prefix)
    (setq local (caar mm))
    (setq signature (cdar mm))
    (setq fn (cadr mm))
    (unless (eq local '|interpOnly|)
      (setq varPart (unless giveVariableIfNil (cons " of" (|bright| v))))
      (setq prefix
        (cons '|' Compiled function type| (append varPart (cons '|: | nil))))
      (sayBrightly (concat '| Declared type or mode| varPart '|: |
        (prefix2String mode))))
    (mapcar #'(lambda (x) (g v x giveVariableIfNil)) val)))

——

defun displayMode

[bright p??]
[fixObjectForPrinting p727]
sayBrightly p??
concat p1197
[prefix2String p??]

—— defun displayMode ——

(defun displayMode (v mode giveVariableIfNil)
(let (varPart)
  (when mode
    (unless giveVariableIfNil
      (setq varPart (cons '| of| (|bright| (|fixObjectForPrinting| v))))
      (|sayBrightly| (|concat| prefix (|formatSignature| signature)))
    (mapcar #'(lambda (x) (g v x giveVariableIfNil)) val)))

——
25.2. FUNCTIONS

(defun Split into tokens delimited by spaces

(stripSpaces p743)

— defun dumbTokenize —

(defun |dumbTokenize| (str)
  (let (inString token (tokenStart 0) previousSpace tokenList)
    (dotimes (i (1- (|#| str)))
      (cond
        ((char= (elt str i) #") ; don't split strings
         (setq inString (null inString))
         (setq previousSpace nil))
        ((and (char= (elt str i) #\space) (null inString))
         (unless previousSpace
           (setq token (|stripSpaces| (subseq str tokenStart i)))
           (setq tokenList (cons token tokenList))
           (setq tokenStart (1+ i))
           (setq previousSpace t)))
        (t
         (setq previousSpace nil)))
    (setq tokenList (cons (|stripSpaces| (subseq str tokenStart)) tokenList))
    (nreverse tokenList)))

—

(defun Convert string tokens to their proper type

(isIntegerString p740)

— defun tokTran —

(defun |tokTran| (tok)
  (let (tmp)
    (if (stringp tok)
      (cond
        ((eql (|#| tok) 0) nil)
        ((setq tmp (|isIntegerString| tok)) tmp)
        ((char= (elt tok 0) #" ) (subseq tok 1 (1- (|#| tok))))
        (t (intern tok)))
    tok))

—
defun Is the argument string an integer?

---

(defun isIntegerString 0 ---

(defun isIntegerString (tok)
  (multiple-value-bind (int len) (parse-integer tok :junk-allowed t)
    (when (and int (= len (length tok))) int)))

------

defun Handle parsed system commands

dumbTokenize p739
parseSystemCmd p740
tokTran p739
systemCommand p720

---

(defun handleParsedSystemCommands ---

(defun handleParsedSystemCommands (unabr optionList)
  (declare (ignore unabr))
  (let (restOptionList parcmd trail)
    (setq restOptionList (mapcar #'dumbTokenize (cdr optionList)))
    (setq parcmd (parseSystemCmd (car optionList)))
    (setq trail
      (mapcar #'(lambda (opt)
        (mapcar #'(lambda (tok) (tokTran tok)) opt)) restOptionList))
    (systemCommand (cons parcmd trail))))

------

defun Parse a system command

tokTran p739
stripSpaces p743
parseFromString p273
dumbTokenize p739

--- defun parseSystemCmd ---

(defun parseSystemCmd (opt)
  (let (spaceIndex)
    (if (setq spaceIndex (search " " opt))

defun Get first word in a string
[subseq p??] [stringSpaces p??]

— defun getFirstWord —

(defun getFirstWord (string)
  (let (spaceIndex)
    (setq spaceIndex (search " " string))
    (if spaceIndex
      (stripSpaces (subseq string 0 spaceIndex))
      string)))

defun Unabbreviate keywords in commands
[selectOption p751] [selectOptionLC p751] [commandsForUserLevel p720] [$systemCommands p715] [$currentLine p??] [$syscommands p716] [line p??]

— defun unAbbreviateKeyword —

(defun unAbbreviateKeyword (x)
  (let (xp)
    (declare (special $systemCommands $currentLine $syscommands line))
    (setq xp (selectOptionLC x $syscommands 'commandErrorIfAmbiguous))
    (cond
      ((null xp)
        (setq xp '|system|)
      (setq line (concat "")system " (substring line 1 (1- (# line)))))
    (setq |$currentLine| line)))
defun The command is ambiguous error

(commandAmbiguityError p724)
(oldline p721)
(line p77)

— defun commandErrorIfAmbiguous —

(defun commandErrorIfAmbiguous | x u |
  (declare (special $oldline line))
  (when u
    (setq $oldline line)
    (commandAmbiguityError |command| x u)))

— defun handleNoParseCommands —

(defun handleNoParseCommands | unab string |
  (let (spaceindex funname)
    (setq string (stripSpaces string))
    (setq spaceindex (search " " string))
    (cond
      ((eq unab |lisp|)
        (if spaceindex
          (nplisp (stripLisp string))
          (sayKeyedMsg "Your argument list is not valid." nil)))
      ((eq unab |boot|)
        (if spaceindex
          (npboot (subseq string (1+ spaceindex)))))}
25.2. FUNCTIONS

(defun Remove the spaces surrounding a string

TPDHERE: This should probably be a macro or eliminated
— defun stripSpaces 0 —

(defun |stripSpaces| (|str|)
  (string-trim '(#\space) |str|))

 Technologies

— defun.stripLisp 0 —

(defun |stripLisp| (|str|)
  (if (string= (subseq |str| 0 4) "lisp")
    (subseq |str| 4)
    |str|))
defun Handle the )lisp command

|$ans p??|

— defun nplisp 0 —

(defun nplisp (str)
  (declare (special |$ans|))
  (setq |$ans| (eval (read-from-string str)))
  (format t "Value = ~S\%" |$ans|))

—

defun The )boot command is no longer supported

TDDHERE: Remove all boot references from top level

— defun npboot 0 —

(defun npboot (str)
  (declare (ignore str))
  (format t "The )boot command is no longer supported\%"))

—

defun Handle the )system command

Note that unAbbreviateKeyword returns the word “system” for unknown words so we have to search for this case. This complication may never arrive in practice.

[sayKeyedMsg p27]

— defun npsystem —

(defun npsystem (unab str)
  (let (spaceIndex sysPart)
    (setq spaceIndex (search " " str))
    (cond
      ((null spaceIndex) (|sayKeyedMsg| "Unknown system command: %1" (list str)))
      (t
       (setq sysPart (subseq str 0 spaceIndex))
       (if (search sysPart (string unab))
        (obey (subseq str (1+ spaceIndex)))
        (|sayKeyedMsg| "Unknown system command: %1" (list sysPart)))))

—
defun Handle the )synonym command

(defun npProcessSynonym p745)

— defun npsynonym —

(defun npsynonym (unab str)
 (declare (ignore unab))
 (npProcessSynonym str))

—

defun Handle the synonym system command

(defun npProcessSynonym)

(defun npProcessSynonym (str)
 (let (pair)
 (declare (special $CommandSynonymAlist))
 (if (= (length str) 0)
 (printSynonyms nil)
 (progn
 (setq pair (processSynonymLine str))
 (if $CommandSynonymAlist
 (putalist $CommandSynonymAlist (car pair) (cdr pair))
 (setq $CommandSynonymAlist (cons pair nil))))
 (terminateSystemCommand)))

—

defun printSynonyms

(defun npProcessSynonym (str)
 (let (pair)
 (declare (special $CommandSynonymAlist))
 (if (= (length str) 0)
 (printSynonyms nil)
 (progn
 (setq pair (processSynonymLine str))
 (if $CommandSynonymAlist
 (putalist $CommandSynonymAlist (car pair) (cdr pair))
 (setq $CommandSynonymAlist (cons pair nil))))
 (terminateSystemCommand)))

—

defun printSynonyms

(defun npProcessSynonym (str)
 (let (pair)
 (declare (special $CommandSynonymAlist))
 (if (= (length str) 0)
 (printSynonyms nil)
 (progn
 (setq pair (processSynonymLine str))
 (if $CommandSynonymAlist
 (putalist $CommandSynonymAlist (car pair) (cdr pair))
 (setq $CommandSynonymAlist (cons pair nil))))
 (terminateSystemCommand)))

—
--- defun printSynonyms ---

(defun printSynonyms (patterns)
  (let ((ls t1))
    (declare (special \$CommandSynonymAlist \$linelength))
    (format t "\n",',,'-:@<\n" System Command Synonyms ")
    (setq ls
      (filterListOfStringsWithFn patterns
        (do ((t2 (synonymsForUserLevel \$CommandSynonymAlist) (cdr t2)))
          (if (null t2) (nreverse0 t1)
            (push (cons (princ-to-string (caar t2)) (cdar t2)) t1))
          #'car))
      ([printLabelledList] ls "user" "synonyms " patterns)))

---

defun Print a list of each matching synonym

The prefix goes before each element on each side of the list, eg, "\n")"

[sayMessage p] [blankList p] [substring p] [entryWidth p] [sayBrightly p] [concat p] [fillerSpaces p]

--- defun printLabelledList ---

(defun printLabelledList (ls label1 label2 prefix patterns)
  (let ((comm syn wid)
    (if (null ls)
      (if (null patterns)
        ([sayMessage] (list " No " label1 "-defined " label2 " in effect.")))
        ([sayMessage] "(" " No " ,label1 "-defined " ,label2 " satisfying patterns:" |%\| " " ,@ append ([blankList] patterns) (list nil)))))
    (progn
      (when patterns
        ([sayMessage] "(" ,label1 "-defined " ,label2 " satisfying patterns:" |%\| " "
          ,@ append ([blankList] patterns) (list nil)))))
      (do ((t1 ls (cdr t1)))
        ((atom t1) nil)
        (setq sym (caar t1)))
25.2. FUNCTIONS

(defun |sayBrightly|)

(defun |fillerSpaces|)

(defun |concat|)

(defun |substring|)

(defun |max|)

(defun |sayBrightly|)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)

(defun ||)
defvar $InitialCommandSynonymAlist

Axiom can create “synonyms” for commands. We create an initial table of synonyms which are in common use.

--- initvars ---

(defun axiomVersion ()
  (declare (special *build-version* *yearweek*))
  (concatenate 'string "Axiom " *build-version* " built on " *yearweek*))

--- postvars ---

(eval-when (eval load)
  (setq $InitialCommandSynonymAlist
    '( (? . "what commands")
      (ap . "what things")
    ))
25.2. FUNCTIONS

|apr| . "what things"
|apropos| . "what things"
|cache| . "set functions cache"
|cl| . "clear"
|cms| . "system"
|co| . "compiler"
|d| . "display"
|dep| . "display dependents"
|dependents| . "display dependents"
|e| . "edit"
|expose| . "set expose add constructor"
|fns| . "exec spadfn"
|fortran| . "set output fortran"
|h| . "help"
|hd| . "system hypertex &"
|kclam| . "boot clearClams ( )"
|killcaches| . "boot clearConstructorAndLisplibCaches ( )"
|prompt| . "set message prompt"
|recurrence| . "set functions recurrence"
|restore| . "history )restore"
|save| . "history )save"
|startGraphics| . "system $AXIOM/lib/viewman &"
|startNAGLink| . "system $AXIOM/lib/nagman &"
|stopGraphics| . "lisp ([sockSendSignal] 2 15)"
|stopNAGLink| . "lisp ([sockSendSignal] 8 15)"
|time| . "set message time"
|type| . "set message type"
|unexpose| . "set expose drop constructor"
|version| . "lisp (axiomVersion)"
|w| . "what"
|wc| . "what categories"
|wd| . "what domains"
|who| . "lisp (pprint credits)"
|wp| . "what packages"
|ws| . "what synonyms"

defvar $CommandSynonymAlist

The actual list of synonyms is initialized to be the same as the above initial list of synonyms. The user synonyms that are added during a session are pushed onto this list for later lookup.

-- initvars --

(defvar |$CommandSynonymAlist| nil)
CHAPTER 25. SYSTEM COMMAND HANDLING

---

-- postvars --

(eval-when (eval load)
 (setq $CommandSynonymAlist (copy-alist $InitialCommandSynonymAlist)))

---

defun ncloopCommand

The $systemCommandFunction is set in SpadInterpretStream to point to the function InterpExecuteSpadSystemCommand. The system commands are handled by the function in the “hook” variable $systemCommandFunction which has the default function InterpExecuteSpadSystemCommand. Thus, when a system command is entered this function is called.

The only exception is the )include function which inserts the contents of a file inline in the input stream. This is useful for processing )read of input files.

[ncloopPrefix? p750]
[ncloopInclude1 p861]
[$systemCommandFunction p??]
[$systemCommandFunction p??]

---

defun ncloopPrefix?

If we find the prefix string in the whole string starting at position zero we return the remainder of the string without the leading prefix.

---

defun ncloopPrefix? 0 --

(defun |ncloopPrefix?| (prefix whole)
\begin{verbatim}
(defun selectOptionLC
  (selectOption p)
  (downcase p)
  (object2Identifier p)
)

(defun selectOption
  (member p)
  (identp p)
  (stringPrefix? p)
  (pname p)
  (qcdr p)
  (qcar p)
)
\end{verbatim}
(cond
  ((and (consp u) (eq (qcdr u) nil)) (progn (setq y (qcar u)) t)) y)
(errorfunction (funcall errorfunction x u))
(t nil)))))))
25.3  \texttt{)abbreviations Command}

abbreviations man page

--- abbreviations.help ---

====================================================================
A.2.  \texttt{)abbreviation}
====================================================================

User Level Required:  compiler

Command Syntax:

- \texttt{)abbreviation query} [nameOrAbbrev]
- \texttt{)abbreviation category} Abbrev Fullname [quiet]
- \texttt{)abbreviation domain} Abbrev Fullname [quiet]
- \texttt{)abbreviation package} Abbrev Fullname [quiet]
- \texttt{)abbreviation remove} nameOrAbbrev

Command Description:

This command is used to query, set and remove abbreviations for category, domain and package constructors. Every constructor must have a unique abbreviation. This abbreviation is part of the name of the subdirectory under which the components of the compiled constructor are stored. Furthermore, by issuing this command you let the system know what file to load automatically if you use a new constructor. Abbreviations must start with a letter and then be followed by up to seven letters or digits. Any letters appearing in the abbreviation must be in uppercase.

When used with the query argument, this command may be used to list the name associated with a particular abbreviation or the abbreviation for a constructor. If no abbreviation or name is given, the names and corresponding abbreviations for all constructors are listed.

The following shows the abbreviation for the constructor List:

\texttt{)abbreviation query List}

The following shows the constructor name corresponding to the abbreviation NNI:

\texttt{)abbreviation query NNI}

The following lists all constructor names and their abbreviations.

\texttt{)abbreviation query}
To add an abbreviation for a constructor, use this command with category, domain or package. The following add abbreviations to the system for a category, domain and package, respectively:

)`abbreviation domain  SET Set
)`abbreviation category COMPCAT ComplexCategory
)`abbreviation package  LIST2MAP ListToMap

If the `)quiet` option is used, no output is displayed from this command. You would normally only define an abbreviation in a library source file. If this command is issued for a constructor that has already been loaded, the constructor will be reloaded next time it is referenced. In particular, you can use this command to force the automatic reloading of constructors.

To remove an abbreviation, the `remove` argument is used. This is usually only used to correct a previous command that set an abbreviation for a constructor name. If, in fact, the abbreviation does exist, you are prompted for confirmation of the removal request. Either of the following commands will remove the abbreviation `VECTOR2` and the constructor name `VectorFunctions2` from the system:

)`abbreviation remove VECTOR2
)`abbreviation remove VectorFunctions2

Also See:
- `)compile`

---

defun abbreviations

[abbreviationsSpad2Cmd p754]

— defun abbreviations —

(defun |abbreviations| (l)
  (|abbreviationsSpad2Cmd| l))

---

defun abbreviationsSpad2Cmd

[listConstructorAbbreviations p756]
[abbreviation? p757]
25.3. ABBREVIATIONS COMMAND

(defun abbreviationsSpad2Cmd (arg)
  (let (abopts quiet opt key type constructor t2 a b al)
    (declare (special $options))
    (if (null arg)
      (helpSpad2Cmd '((abbreviations)))
      (progn
        (setq abopts '([query] [domain] [category] [package] [remove]))
        (setq quiet nil)
        (do ((t0 $options (cdr t0)) (t1 nil)
            ((or (atom t0)
                (progn (setq t1 (car t0)) nil)
                (progn (setq opt (car t1)) t1) nil)) nil)
          (setq opt (selectOptionLC opt '(quiet) 'optionError))
          (when (eq opt 'quiet) (setq quiet t)))
      (when
        (and (consp arg)
          (progn
            (setq opt (qcar arg))
            (setq al (qcdr arg))
            t))
        (setq key (opOf (car al)))
        (setq type (selectOptionLC opt abopts 'optionError))
        (cond ((eq type 'query)
          (cond
            ((null al) (listConstructorAbbreviations))
            ((setq constructor (abbreviation? key))
              (abbQuery constructor))
            (t (abbQuery key)))
            ((eq type 'remove))))
(deldatabase key 'abbreviation))
((oddp (size al))
 (|sayKeyedMsg|
  (format nil
   "%1 must be followed by an alternating list of abbreviation(s) ~
   and name(s). Issue )abbrev ? for more information.")
  (list type)))
(t
 (do () (nil nil)
  (seq
   (exit
    (cond
     ((null al) (return '|fromLoop|))
     (t
      (setq t2 al)
      (setq a (car t2))
      (setq b (cadr t2))
      (setq al (cddr t2))
      (|mkUserConstructorAbbreviation| b a type)
      (setdatabase b 'abbreviation a)
      (setdatabase b 'constructorkind type))))
    (unless quiet
      (|sayKeyedMsg| "%1 abbreviates % %2 %3 %"
       (list a type (|opOf| b))))))))

(defun listConstructorAbbreviations
  [upcase p1206]
  [queryUserKeyedMsg p??]
  [string2id-n p??]
  [whatSpad2Cmd p1091]
  [sayKeyedMsg p27]

  — defun listConstructorAbbreviations —

  (defun list ConstructorAbbreviations
    ()
    (let (x)
      (setq x
        (upcase
          (|queryUserKeyedMsg|
            (format nil
              "You have requested that all abbreviations be displayed. As there are ~
   several hundred abbreviations, please confirm your request by ~
   typing y or yes and then pressing Enter :")
            nil)))))
(if (member (string2id-n x 1) '(Y YES))
  (progn
    (|whatSpad2Cmd| '(|categories|))
    (|whatSpad2Cmd| '(|domains|))
    (|whatSpad2Cmd| '(|packages|))
    (|sayKeyedMsg|
      (format nil
        "Since you did not respond with y or yes the list of abbreviations ~
         will not be displayed."
        nil))))
25.4 \texttt{\textbackslash )boot Command}

\texttt{boot man page}

\begin{verbatim}
— boot.help —

====================================================================
A.3. \texttt{\textbackslash boot}
====================================================================

User Level Required: development

Command Syntax:
- \texttt{\textbackslash boot bootExpression}

Command Description:

This command is used by AXIOM system developers to execute expressions written in the BOOT language. For example,

\texttt{\textbackslash boot times3(x) == 3*x}

creates and compiles the Lisp function \texttt{‘times3’} obtained by translating the BOOT code.

Also See:
- \texttt{\textbackslash fin}
- \texttt{\textbackslash lisp}
- \texttt{\textbackslash set}
- \texttt{\textbackslash system}

\end{verbatim}

\textsuperscript{1}

This command is in the list of \texttt{\$noParseCommands 25.1} which means that its arguments are passed verbatim. This will eventually result in a call to the function \texttt{handleNoParseCommands 25.2}

\textsuperscript{1} “\texttt{fin}” (25.20 p 808) “\texttt{lisp}” (25.27 p 867) “\texttt{set}” (25.51 p 1013) “\texttt{system}” (25.56 p 1041)
25.5  \textit{)browse Command}

\textit{browse man page}

--- browse.help ---

User Level Required: development

Command Syntax:

\texttt{)browse}

Command Description:

This command is used by Axiom system users to start the Axiom top level loop listening for browser connections.

---

25.6  Overview

The Axiom book on the help browser is a complete rewrite of the hyperdoc mechanism. There are several components that were needed to make this function. Most of the web browser components are described in bookvol11.pamphlet. This portion describes some of the design issues needed to support the interface.

The \texttt{axServer} command takes a port (defaulting to 8085) and a program to handle the browser interaction (defaulting to multiServ). The \texttt{axServer} function opens the port, constructs the stream, and passes the stream to multiServ. The multiServ loop processes one interaction at a time.

So the basic process is that the Axiom \texttt{\textasciitilde)browse} command opens a socket and listens for http requests. Based on the type of request (either ‘GET’ or ‘POST’) and the content of the request, which is one of:

- command - algebra request/response
- lispcall - a lisp s-expression to be evaluated
- showcall - an Axiom \texttt{\textasciitilde)show} command

the multiServ function will call a handler function to evaluate the command line and construct a response. GET requests result in a new browser page. POST requests result in an inline result.

Most responses contain the fields:
CHAPTER 25. SYSTEM COMMAND HANDLING

- stepnum - this is the Axiom step number
- command - this is the original command from the browser
- algebra - this is the Axiom 2D algebra output
- mathml - this is the MathML version of the Axiom algebra
- type - this is the type of the Axiom result

25.7 Browsers, MathML, and Fonts

This work has the Firefox browser as its target. Firefox has built-in support for MathML, javascript, and XMLHttpRequests. More details are available in bookvol11.pamphlet but the very basic machinery for communication with the browser involves a dance between the browser and the multiServ function (see the axserver.spad.pamphlet).

In particular, a simple request is embedded in a web page as:

```html
<ul>
  <li>
    <input type="submit" id="p3" class="subbut"
      onclick="makeRequest('p3');"
      value="sin(x)" />
    <div id="ansp3"><div></div></div>
  </li>
</ul>
```

which says that this is an html “input” field of type “submit”. The CSS display class is “subbut” which is of a different color than the surrounding text to make it obvious that you can click on this field. Clickable fields that have no response text are of class “noresult”.

The javascript call to “makeRequest” gives the “id” of this input field, which must be unique in the page, as an argument. In this case, the argument is ‘p3’. The “value” field holds the display text which will be passed back to Axiom as a command.

When the result arrives the “showanswer” function will select out the mathml field of the response, construct the “id” of the html div to hold the response by concatenating the string “ans” (answer) to the “id” of the request resulting, in this case, as “ansp3”. The “showanswer” function will find this div and replace it with a div containing the mathml result.

The “makeRequest” function is:

```javascript
function makeRequest(arg) {
    http_request = new XMLHttpRequest();
    var command = commandline(arg);
    //alert(command);
    http_request.open('POST', '127.0.0.1:8085', true);
    http_request.onreadystatechange = handleResponse;
    http_request.send(command);
}
```
http_request.setRequestHeader('Content-Type', 'text/plain');
http_request.send("command="+command);
return(false);

It contains a request to open a local server connection to Axiom, sets “handleResponse” as the function to call on reply, sets up the type of request, fills in the command field, and sends off the http request.

When a response is received, the “handleResponse” function checks for the correct reply state, strips out the important text, and calls “showanswer”.

function handleResponse() {
  if (http_request.readyState == 4) {
    if (http_request.status == 200) {
      showanswer(http_request.responseText,'mathAns');
    } else {
      alert('There was a problem with the request.'+ http_request.statusText);
    }
  }
}

See bookvol11.pamphlet for further details.

25.8 The axServer/multiServ loop

The basic call to start an Axiom browser listener is:

)set message autoload off
)set output mathml on
axServer(8085,multiServ)$AXSERV

This call sets the port, opens a socket, attaches it to a stream, and then calls “multiServ” with that stream. The “multiServ” function loops serving web responses to that port.

25.9 The )browse command

In order to make the whole process cleaner the function “)browse” handles the details. This code creates the command-line function for )browse

The browse function does the internal equivalent of the following 3 command line statements:

)set message autoload off
)set output mathml on
axServer(8085,multiServ)$AXSERV
which causes Axiom to start serving web pages on port 8085

For those unfamiliar with calling algebra from lisp there are a few points to mention.

The loadLib needs to be called to load the algebra code into the image. Normally this is automatic but we are not using the interpreter so we need to do this “by hand”.

Each algebra file contains a ”constructor function” which builds the domain, which is a vector, and then caches the vector so that every call to the constructor returns an EQ vector, that is, the same vector. In this case, we call the constructor [AxiomServer]

The axServer function was mangled internally to \texttt{|AXSERV;axServer;IMV;2|}. The multiServ function was mangled to \texttt{|AXSERV;multiServ;SeV;3|}. Note well that if you change axserver.spad these names might change which will generate the error message along the lines of:

\begin{verbatim}
  System error:
  The function \texttt{|AXSERV;axServer;IMV;2|} is undefined.
\end{verbatim}

To fix this you need to look at int/algebra/AXSERV.nrlib/code.lsp and find the new mangled function name. A better solution would be to dynamically look up the surface names in the domain vector.

Each Axiom function expects the domain vector as the last argument. This is not obvious from the call as the interpreter supplies it. We must do that “by hand”.

We don’t call the multiServ function. We pass it as a parameter to the axServer function. When it does get called by the SPADCALL macro it needs to be a lisp pair whose car is the function and whose cdr is the domain vector. We construct that pair here as the second argument to axServer. The third, hidden, argument to axServer is the domain vector which we supply “by hand”.

The socket can be supplied on the command line but defaults to 8085. Axiom supplies the arguments as a list.

\begin{verbatim}
[set p1013]
[loadLib p1181]
[AxiomServer p??]
[AXSERV;axServer;IMV;2 p??]
\end{verbatim}

\begin{verbatim}
— defun browse —
\end{verbatim}

\begin{verbatim}
defun browse (socket)
  (let (axserv browser)
    (if socket
      (setq socket (car socket))
      (setq socket 8085))
      (setq axserv (AxiomServer))
      (setq browser

\end{verbatim}
Now we have to bolt it into Axiom. This involves two lookups.
We create the lisp pair

```
(|browse| . |development|)
```

and cons it into the $systemCommands command table. This allows the command to be executed in development mode. This lookup decides if this command is allowed. It also has the side-effect of putting the command into the $SYSCOMMANDS variable which is used to determine if the token is a command.

25.10 The server support code
25.11 \texttt{\textbackslash{}cd} Command

\texttt{cd man page}

--- \texttt{cd\_help} ---

\begin{verbatim}
A.4. \texttt{\textbackslash{}cd}
\end{verbatim}

User Level Required: interpreter

Command Syntax:

- \texttt{\textbackslash{}cd directory}

Command Description:

This command sets the AXIOM working current directory. The current directory is used for looking for input files (for \texttt{\textbackslash{}read}), AXIOM library source files (for \texttt{\textbackslash{}compile}), saved history environment files (for \texttt{\textbackslash{}history\textbackslash{}restore}), compiled AXIOM library files (for \texttt{\textbackslash{}library}), and files to edit (for \texttt{\textbackslash{}edit}). It is also used for writing spool files (via \texttt{\textbackslash{}spool}), writing history input files (via \texttt{\textbackslash{}history\textbackslash{}write}) and history environment files (via \texttt{\textbackslash{}history\textbackslash{}save}), and compiled AXIOM library files (via \texttt{\textbackslash{}compile}).

If issued with no argument, this command sets the AXIOM current directory to your home directory. If an argument is used, it must be a valid directory name. Except for the \texttt{\textbackslash{}}} at the beginning of the command, this has the same syntax as the operating system \texttt{cd} command.

Also See:

- \texttt{\textbackslash{}compile}
- \texttt{\textbackslash{}edit}
- \texttt{\textbackslash{}history}
- \texttt{\textbackslash{}library}
- \texttt{\textbackslash{}read}
- \texttt{\textbackslash{}spool}

\footnote{\texttt{\textbackslash{}edit} (25.19 p 804) \texttt{\textbackslash{}history} (25.23 p 821) \texttt{\textbackslash{}library} (34.1 p 1159) \texttt{\textbackslash{}read} (25.31 p 875) \texttt{\textbackslash{}spool} (25.53 p 1033)}
25.12  )clear Command

This command is used to remove function and variable declarations, definitions and values from the workspace. To empty the entire workspace and reset the step counter to 1, issue

)clear all

To remove everything in the workspace but not reset the step counter, issue

)clear properties all

To remove everything about the object x, issue

)clear properties x

To remove everything about the objects x, y and f, issue

)clear properties x y f

The word properties may be abbreviated to the single letter ‘‘p’’.

)clear p all
)clear p x
)clear p x y f
All definitions of functions and values of variables may be removed by either

)clear value all
)clear v all

This retains whatever declarations the objects had. To remove definitions and
values for the specific objects x, y and f, issue

)clear value x y f
)clear v x y f

To remove the declarations of everything while leaving the definitions and
values, issue

)clear mode all
)clear m all

To remove declarations for the specific objects x, y and f, issue

)clear mode x y f
)clear m x y f

The )display names and )display properties commands may be used to see what
is currently in the workspace.

The command

)clear completely

does everything that )clear all does, and also clears the internal system
function and constructor caches.

Also See:
- )display
- )history
- )undo

---

defvar $clearOptions

--- initvars ---

---

3 “display” (25.18 p 790) “history” (25.23 p 821) “undo” (25.60 p 1072)
(defvar $clearOptions '(|modes| |operations| |properties| |types| |values|))

---

defun clear

clearSpad2Cmd p767

---

defun clear

(defun |clear| (l)
   (|clearSpad2Cmd| l))

---

defvar $clearExcept

---

initvars

(defvar |$clearExcept| nil)

---

defun clearSpad2Cmd

TPDHERE: Note that this function also seems to parse out )except )completely and )scaches
which don’t seem to be documented. [selectOptionLC p751]

|sayKeyedMsg p27|
|clearCmdAll p771|
|clearCmdCompletely p770|
|clearCmdSortedCaches p769|
|clearCmdExcept p772|
|clearCmdParts p773|
|updateCurrentInterpreterFrame p16|
|$clearExcept p767|
|$options p??|
|$clearOptions p766|

---

defun clearSpad2Cmd
(defun |clearSpad2Cmd| (l)
  (let (($clearExcept| opt optlist arg)
    (declare (special |$clearExcept| |$options| |$clearOptions|)))
    (cond
      (|$options|
        (setq |$clearExcept|
          (prog (t0)
            (setq t0 t)
            (return
              (do ((t1 nil (null t0))
                    (t2 |$options| (cdr t2))
                    (t3 nil))
                ((or t1
                    (atom t2)
                    (progn (setq t3 (car t2)) nil)
                    (progn (progn (setq opt (car t3)) t3) nil))
                 t0)
              (setq t0
                (and t0
                  (eq
                    (|selectOptionLC| opt '(|except|) '|optionError|)
                    '|except|))))))
    (cond
      ((null l)
        (setq optlist
          (prog (t4)
            (setq t4 nil)
            (return
              (do ((t5 |$clearOptions| (cdr t5)) (x nil))
                  ((or (atom t5) (progn (setq x (car t5)) nil)) t4)
                (setq t4 (append t4 `(|%l| " " ,x)))))
          (|sayKeyedMsg|
            (format nil
              "Use )clear all to clear everything in the workspace. Use )clear ~
                completely to clear everything in the workspace and internal ~
                tables. Other )clear keyword arguments are %1 %l or abbreviations ~
                thereof. Issue )clear ? for more information.")
            (list optlist))))
    (t
      (setq arg
        (|selectOptionLC| (car l) '(|all| |completely| |scaches|) nil))
      (cond
        ((eq arg '|all|) (|clearCmdAll|))
        ((eq arg '|completely|) (|clearCmdCompletely|))
        ((eq arg '|scaches|) (|clearCmdSortedCaches|))
        (|$clearExcept| (|clearCmdExcept| l))
        (t
          (|clearCmdParts| l)
          (|updateCurrentInterpreterFrame|)))))
25.12. CLEAR COMMAND

---

(defun clearCmdSortedCaches
  [compiledLookupCheck p769]
  [spadcall p?]
  [$lookupDefaults p?]
  [$Void p645]
  [$ConstructorCache p?]

  — defun clearCmdSortedCaches —

  (defun clearCmdSortedCaches ()
    (let (|$lookupDefaults| domain pair)
      (declare (special $lookupDefaults Void ConstructorCache))
      (do ((t0 (hget ConstructorCache 'SortedList) (cdr t0))
           (t1 nil)
           ((or (atom t0)
                (progn
t0 (car t0))
            (setq domain (cddr t1))
           )
           nil)
       (setq pair (compiledLookupCheck 'clearCache (list Void) domain))
       (spadcall pair))))

---

defun compiledLookupCheck
  [compiledLookup p1185]
  [keyedSystemError p?]
  [formatSignature p?]

  — defun compiledLookupCheck —

  (defun compiledLookupCheck (op sig dollar)
    (let (fn)
      (setq fn (compiledLookup op sig dollar))
      (cond
        ((and (null fn) (eq op '^))
         (setq fn (compiledLookup '** sig dollar)))
        ((and (null fn) (eq op '**))
         (setq fn (compiledLookup '~ sig dollar)))
        (t nil))
      (cond
((null fn)
   |keyedSystemError|
   "The function %1 with signature %2 is missing from domain %3"
   (list op (|formatSignature| sig) (elt dollar 0)))
   (t fn)))))

defvar $functionTable

— initvars —

(defvar |$functionTable| nil)

---

defun clearCmdCompletely

[clearCmdAll p771]
sayKeyedMsg p27
[clearClams p??]
[clearConstructorCaches p??]
[reclaim p263]
|$localExposureData| p98
|xdatabase p??]
|CatOfCatDatabase p??]
|DomOfCatDatabase p??]
|JoinOfCatDatabase p??]
|JoinOfDomDatabase p??]
|attributeDb p??]
|functionTable p770]
|existingFiles p??]
|$localExposureDataDefault| p99]

— defun clearCmdCompletely —

(defun |clearCmdCompletely| ()
   (declare (special |$localExposureData| |xdatabase| |CatOfCatDatabase|
   |DomOfCatDatabase| |JoinOfCatDatabase| |JoinOfDomDatabase|
   |attributeDb| |functionTable| |existingFiles|
   |localExposureDataDefault|))
   (|clearCmdAll|)
   (setq |$localExposureData| (copy-seq |$localExposureDataDefault|)))
(setq $xdatabase| nil)
(setq $CatOfCatDatabase| nil)
(setq $DomOfCatDatabase| nil)
(setq $JoinOfCatDatabase| nil)
(setq $JoinOfDomDatabase| nil)
(setq $attributeDb| nil)
(setq $functionTable| nil)

(\|sayKeyedMsg\| "All )browse facility databases have been cleared." nil)
((clearClams))
((clearConstructorCaches))
(setq $existingFiles| (make-hash-table :test #\'equal))
(\|sayKeyedMsg\|
 "Internally cached functions and constructors have been cleared." nil)
(reclaim)
(\|sayKeyedMsg\| ")clear completely is finished." nil))

----------

defun clearCmdAll

[clearCmdSortedCaches p769]
[untraceMapSubNames p1059]
[resetInCoreHist p830]
deleteFile p1193]
[histFileName p820]
[updateCurrentInterpreterFrame p16]
clearMacroTable p772]
sayKeyedMsg p27]
[frameRecord p35]
[previousBindings p36]
[variableNumberAlist p77]
[interactiveFrame p23]
[useInternalHistoryTable p819]
[internalHistoryTable p32]
[frameMessages p945]
[interpreterFrameName p23]
[currentLine p77]

defun clearCmdAll

(defun clearCmdAll ()
 (declare (special |$frameRecord| |$previousBindings| |$variableNumberAlist|
              |$interactiveFrame| |$useInternalHistoryTable| |$internalHistoryTable|
              |$frameMessages| |$interpreterFrameName| |$currentLine|))
 (clearCmdSortedCaches))
(setq |$frameRecord| nil)
(setq $previousBindings| nil)
(setq |$variableNumberAlist| nil)
(untraceMapSubNames| /tracenames)
(setq |$InteractiveFrame| (list (list nil)))
(resetInCoreHist|)
(when |$useInternalHistoryTable|
    (setq |$internalHistoryTable| nil)
    (deleteFile| (histFileName|)))
(setq |$IOindex| 1)
(updateCurrentInterpreterFrame|)
(setq |$currentLine| "clear all")
(clearMacroTable|)
(when |$frameMessages|
    (sayKeyedMsg|
        (format nil
            "All user variables and function definitions have been cleared in ~
            the current frame ( %1 ).")
            (list |$interpreterFrameName|))
    (sayKeyedMsg|
        "All user variables and function definitions have been cleared." nil)))

defun clearMacroTable

[pfMacros p324]

defun clearCmdExcept

[clearCmdParts p773]
[clearOptions p766]

(defun |clearCmdExcept| (arg)
defun clearCmdParts

[selectOptionLC p751]  
[pname p1195]  
[types p??]  
[modes p??]  
[values p??]  
[boot-equal p??]  
[assocleft p??]  
[remdup p??]  
[assoc p??]  
isMap p??]  
[get p??]  
[exit p??]  
[untraceMapSubNames p1059]  
[seq p??]  
[recordOldValue p833]  
[recordNewValue p833]  
[deleteAssoc p??]  
[sayKeyedMsg p27]  
[getParserMacroNames p725]  
[getInterpMacroNames p??]  
[clearDependencies p??]  
[member p1198]  
[clearParserMacro p725]  
[sayMessage p??]  
[fixObjectForPrinting p727]  
[$e p247]  
[$InteractiveFrame p23]  
[$clearOptions p766]

— defun clearCmdParts —
(setq option (intern (pname option)))
(setq option
  (case option
    ([types] '[:model])
    ([modes] '[:model])
    ([values] '[:value])
    (t option)))
(if (null vl)
  (sayKeyedMsg)
  (format nil
    "After the property you wish to clear you must give one or more ~
    identifiers or specify all to clear that property from everything.")
  nil)
(progn
  (setq pmacs (getParserMacroNames))
  (setq imacs (getInterpMacroNames))
  (cond
    ((boot-equal vl '[:all])
    (setq vl (assocleft (caar $InteractiveFrame))))
    (setq vl (remdup (append vl pmacs))))
  (setq $e| $InteractiveFrame|
  (do ((t0 vl (cdr t0)) (x nil))
    ((or (atom t0) (progn (setq x (car t0)) nil)) nil)
      (clearDependencies| x t)
    (when (and (eq option '[:properties]) (member| x pmacs))
        (clearParserMacro| x))
    (when (and (eq option '[:properties])
      (member| x imacs)
      (null (member| x pmacs)))
      (sayMessage| (cons
        " You cannot clear the definition of the system-defined macro "
        (cons (fixObjectForPrinting| x)
        (cons (intern "." "BOOT") nil))))))
  (cond
    ((setq p1 (assoc| x (caar $InteractiveFrame)))
    (cond
      ([isMap] x)
      (seq
        (cond
          ((setq lm
            (get| x '[:localModemap] $InteractiveFrame)))
          (cond
            ((consp lm)
              (exit (untraceMapSubNames| (cons (cadar lm) nil))))
          (t nil))))
        (dolist (p2 (cdr p1))
          (setq prop (car p2))
          (recordOldValue| x prop (cdr p2)))
    )
    (eq option '[:properties])
    (cond
      ([isMap] x)
      (seq
        (cond
          ((setq lm
            (get| x '[:localModemap] $InteractiveFrame)))
          (cond
            ((consp lm)
              (exit (untraceMapSubNames| (cons (cadar lm) nil))))
          (t nil)))))))
(recordNewValue x prop nil)
(setf (caar $InteractiveFrame)
     (deleteAssoc x (caar $InteractiveFrame))))
((setq p2 (assoc option (cdr p1)))
 (recordOldValue x option (cdr p2))
 (recordNewValue x option nil)
 (rplacd p2 nil))))
nil)
25.13 close Command

close man page

— close.help —

====================================================================
A.5. close
====================================================================

User Level Required: interpreter

Command Syntax:

- )close
- )close quietly

Command Description:

This command is used to close down interpreter client processes. Such processes are started by HyperDoc to run AXIOM examples when you click on their text. When you have finished examining or modifying the example and you do not want the extra window around anymore, issue

)close

to the AXIOM prompt in the window.

If you try to close down the last remaining interpreter client process, AXIOM will offer to close down the entire AXIOM session and return you to the operating system by displaying something like

This is the last AXIOM session. Do you want to kill AXIOM?

Type "y" (followed by the Return key) if this is what you had in mind. Type "n" (followed by the Return key) to cancel the command.

You can use the )quietly option to force AXIOM to close down the interpreter client process without closing down the entire AXIOM session.

Also See:
- )quit
- )pquit

———
defun queryClients

Returns the number of active scratchpad clients

(sockSendInt |$SessionManager| |$QueryClients|)
(sockGetInt |$SessionManager|)

--- defun queryClients ---

(defun |queryClients| ()
  (declare (special |$SessionManager| |$QueryClients|))
  (|sockSendInt| |$SessionManager| |$QueryClients|)
  (|sockGetInt| |$SessionManager|))

---

defun close

(throwKeyedMsg |$SpadServer|)
(sockSendInt |$SessionManager|)
(closeInterpreterFrame)
(selectOptionLC)
(upcase)
(queryUserKeyedMsg |$SpadServer|)
(string2id-n)
(queryClients |$SpadServer|)
($SpadServer |$SessionManager| |$CloseClient| |$currentFrameNum| |$options|)

--- defun close ---

(defun |close| (args)
  (declare (ignore args))
  (let (numClients opt fullopt quiet x)
    (declare (special |$SpadServer| |$SessionManager| |$CloseClient| |$currentFrameNum| |$options|))
    (if (null |$SpadServer|)
      (quit)
      (quit)
(|throwKeyedMsg| "You cannot close this Axiom session." nil))
(progn
  (setq numClients (|queryClients|))
  (cond
    ((> numClients 1)
      (|sockSendInt| |$SessionManager| |$CloseClient|)
      (|sockSendInt| |$SessionManager| |$currentFrameNum|)
      (|closeInterpreterFrame| nil))
    (t
      (do ((t0 |$options| (cdr t0)) (t1 nil))
        ((or (atom t0)
         (progn (setq t1 (car t0)) nil)
         (progn (progn (setq opt (car t1)) t1) nil))
          nil)
      (setq fullopt (|selectOptionLC| opt '(|quiet|) '|optionError|))
      (unless quiet (setq quiet (eq fullopt '|quiet|)))
      (cond
        (quiet
          (|sockSendInt| |$SessionManager| |$CloseClient|)
          (|sockSendInt| |$SessionManager| |$currentFrameNum|)
          (|closeInterpreterFrame| nil))
        (t
          (setq x
            (upcase
              (|queryUserKeyedMsg|
                "This is the last Axiom session. Do you want to kill Axiom?"
                nil)))
          (when (member (string2id-n x 1) '(yes y)) (bye))))))))

|
25.14  )compile Command

compile man page

— compile.help —

====================================================================
A.7.  )compile
====================================================================

User Level Required:  compiler

Command Syntax:

- )compile
- )compile fileName
- )compile fileName.spad
- )compile directory/fileName.spad
- )compile fileName )quiet
- )compile fileName )noquiet
- )compile fileName )break
- )compile fileName )nobreak
- )compile fileName )library
- )compile fileName )nolibrary
- )compile fileName )vartrace
- )compile fileName )constructor nameOrAbbrev

Command Description:

You use this command to invoke the AXIOM library compiler. This
compiles files with file extension .spad with the AXIOM system
compiler. The command first looks in the standard system directories
for files with extension .spad.

Should you not want the )library command automatically invoked, call )compile
with the )nolibrary option. For example,

)compile mycode )nolibrary

By default, the )library system command exposes all domains and categories it
processes. This means that the AXIOM interpreter will consider those domains
and categories when it is trying to resolve a reference to a function.
Sometimes domains and categories should not be exposed. For example, a domain
may just be used privately by another domain and may not be meant for
top-level use. The )library command should still be used, though, so that the
code will be loaded on demand. In this case, you should use the )nolibrary
option on )compile and the )noexpose option in the )library command. For
example,
Once you have established your own collection of compiled code, you may find it handy to use the )dir option on the )library command. This causes )library to process all compiled code in the specified directory. For example,

)library )dir /u/jones/as/quantum

You must give an explicit directory after )dir, even if you want all compiled code in the current working directory processed.

)library )dir .

You can compile category, domain, and package constructors contained in files with file extension .spad. You can compile individual constructors or every constructor in a file.

The full filename is remembered between invocations of this command and )edit commands. The sequence of commands

)compile matrix.spad
)edit
)compile

will call the compiler, edit, and then call the compiler again on the file matrix.spad. If you do not specify a directory, the working current directory (see description of command )cd ) is searched for the file. If the file is not found, the standard system directories are searched.

If you do not give any options, all constructors within a file are compiled. Each constructor should have an )abbreviation command in the file in which it is defined. We suggest that you place the )abbreviation commands at the top of the file in the order in which the constructors are defined. The list of commands serves as a table of contents for the file.

The )library option causes directories containing the compiled code for each constructor to be created in the working current directory. The name of such a directory consists of the constructor abbreviation and the .NRLIB file extension. For example, the directory containing the compiled code for the MATRIX constructor is called MATRIX.NRLIB. The )nolibrary option says that such files should not be created.

The )vartrace option causes the compiler to generate extra code for the constructor to support conditional tracing of variable assignments. (see description of command )trace ). Without this option, this code is suppressed and one cannot use the )vars option for the trace command.

The )constructor option is used to specify a particular constructor to
25.14. \textsc{)compile command}

compile. All other constructors in the file are ignored. The constructor name or abbreviation follows \texttt{)constructor}. Thus either

\texttt{)compile matrix.spad \texttt{)constructor RectangularMatrix}}

or

\texttt{)compile matrix.spad \texttt{)constructor RMATRIX}}

compiles the RectangularMatrix constructor defined in matrix.spad.

The \texttt{)break} and \texttt{)nobreak} options determine what the compiler does when it encounters an error. \texttt{)break} is the default and it indicates that processing should stop at the first error. The value of the \texttt{)set break} variable then controls what happens.

\textbf{Also See:}
\begin{itemize}
  \item \texttt{)abbreviation}
  \item \texttt{)edit}
  \item \texttt{)library}
\end{itemize}

\footnote{\textit{abbreviation} (25.19 p 804) \textit{edit} (34.1 p 1159) \textit{library} (34.1 p 1159) }

\texttt{defvar /editfile}

\texttt{--- initvars ---}

\texttt{(defvar /editfile nil)}
25.15 )copyright Command

copyright man page

— copyright.help —

The term Axiom, in the field of computer algebra software, along with AXIOM and associated images are common-law trademarks. While the software license allows copies, the trademarks may only be used when referring to this project.

Axiom is distributed under terms of the Modified BSD license. Axiom was released under this license as of September 3, 2002. Source code is freely available at: http://savannah.nongnu.org/projects/axiom
Copyrights remain with the original copyright holders. Use of this material is by permission and/or license. Individual files contain reference to these applicable copyrights. The copyright and license statements are collected here for reference.

Portions Copyright (c) 2003- The Axiom Team

The Axiom Team is the collective name for the people who have contributed to this project. Where no other copyright statement is noted in a file this copyright will apply.

Portions Copyright (c) 1991-2002, The Numerical ALgorithms Group Ltd. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of The Numerical ALgorithms Group Ltd. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER
OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Portions Copyright (C) 1989-95 GROUPE BULL

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to
deal in the Software without restriction, including without limitation the
rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
sell copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
GROUPE BULL BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of GROUPE BULL shall not be
used in advertising or otherwise to promote the sale, use or other dealings
in this Software without prior written authorization from GROUPE BULL.

Portions Copyright (C) 2002, Codemist Ltd. All rights reserved.
acn@codemist.co.uk

CCL Public License 1.0

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

(1) Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.

(2) Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.
(3) Neither the name of Codemist nor the names of other contributors may be used to endorse or promote products derived from this software without specific prior written permission.

(4) If you distribute a modified form or either source or binary code
(a) you must make the source form of these modification available to Codemist;
(b) you grant Codemist a royalty-free license to use, modify or redistribute your modifications without limitation;
(c) you represent that you are legally entitled to grant these rights and that you are not providing Codemist with any code that violates any law or breaches any contract.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Portions Copyright (C) 1995-1997 Eric Young (eay@mincom.oz.au) 
All rights reserved.

This package is an SSL implementation written by Eric Young (eay@mincom.oz.au).
The implementation was written so as to conform with Netscapes SSL.

This library is free for commercial and non-commercial use as long as the following conditions are aheared to. The following conditions apply to all code found in this distribution, be it the RC4, RSA, lhash, DES, etc., code; not just the SSL code. The SSL documentation included with this distribution is covered by the same copyright terms except that the holder is Tim Hudson (tjh@mincom.oz.au).

Copyright remains Eric Young’s, and as such any Copyright notices in the code are not to be removed.

If this package is used in a product, Eric Young should be given attribution as the author of the parts of the library used.
This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:
   "This product includes cryptographic software written by Eric Young (eay@mincom.oz.au)"
   The word 'cryptographic' can be left out if the routines from the library being used are not cryptographic related :-).
4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement:
   "This product includes software written by Tim Hudson (tjh@mincom.oz.au)"

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The licence and distribution terms for any publically available version or derivative of this code cannot be changed. i.e. this code cannot simply be copied and put under another distribution licence [including the GNU Public Licence.]

Portions Copyright (C) 1988 by Leslie Lamport.

Portions Copyright (c) 1998 Free Software Foundation, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, distribute with modifications, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE ABOVE COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name(s) of the above copyright holders shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization.

Portions Copyright 1989-2000 by Norman Ramsey. All rights reserved.

Noweb is protected by copyright. It is not public-domain software or shareware, and it is not protected by a ‘‘copyleft’’ agreement like the one used by the Free Software Foundation.

Noweb is available free for any use in any field of endeavor. You may redistribute noweb in whole or in part provided you acknowledge its source and include this COPYRIGHT file. You may modify noweb and create derived works, provided you retain this copyright notice, but the result may not be called noweb without my written consent.

You may sell noweb if you wish. For example, you may sell a CD-ROM including noweb.

You may sell a derived work, provided that all source code for your derived work is available, at no additional charge, to anyone who buys your derived work in any form. You must give permission for said source code to be used and modified under the terms of this license. You must state clearly that your work uses or is based on noweb and that noweb is available free of charge. You must also request that bug reports on your work be reported to you.

Portions Copyright (c) 1987 The RAND Corporation. All rights reserved.


All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the names of Stichting Mathematisch Centrum or CWI not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Portions Copyright (c) Renaud Rioboo and the University Paris 6.

Portions Copyright (c) 2003-2010 Jocelyn Guidry

Portions Copyright (c) 2001-2010 Timothy Daly

---

defun copyright

[obey p??]
[concat p1197]
[getenviron p255]

---

---

defun trademark

---

---

This command is in the list of \texttt{\$noParseCommands 25.1} which means that its arguments are passed verbatim. This will eventually result in a call to the function \texttt{handleNoParseCommands 25.2}
25.16 )credits Command

credits man page
defun credits

[credits p788]

— defun credits 0 —

(defun |credits| ()
 (declare (special credits))
 (mapcar #'(lambda (x) (princ x) (terpri)) creditlist))

This command is in the list of $noParseCommands 25.1 which means that its arguments are passed verbatim. This will eventually result in a call to the function handleNoParseCommands 25.2
25.17  )DESCRIBE COMMAND

describe man page

— describe.help —

=================================================================================
)describe
=================================================================================

User Level Required: interpreter

Command Syntax:

- )describe categoryName
- )describe domainName
- )describe packageName

Command Description:

This command is used to display the comments for the operation, category, domain or package. The comments are part of the algebra source code.

The commands

)describe <categoryName> [internal]
)describe <domainName> [internal]
)describe <packageName> [internal]

will show a properly formatted version of the "Description:" keyword from the comments in the algebra source for the category, domain, or package requested.

If 'internal' is requested, then the internal format of the domain or package is described. Categories do not have an internal representation.

-----

defvar $describeOptions

The current value of $describeOptions is

— initvars —

(defvar $describeOptions '((category| domain| package|))

-----
defun Print comment strings from algebra libraries

This trivial function satisfies the standard pattern of making a user command match the name of the function which implements the command. That command immediately invokes a “Spad2Cmd” version. [describeSpad2Cmd p789]

— defun describe —

(defun describe) (1)
(describeSpad2Cmd 1))

defun describeSpad2Cmd

The describe command prints cleaned-up comment strings from the algebra libraries. It can print strings associated with a category, domain, package, or by operation.

This implements command line options of the form:

)describe categoryName [internal]
)describe domainName [internal]
)describe packageName [internal]

The describeInternal function will either call the “dc” function to describe the internal representation of the argument or it will print a cleaned up version of the text for the ”Description” keyword in the Category, Domain, or Package source code.

[selectOptionLC p751]
[flatten p793]
[cleanline p791]
[getdatabase p1156]
[sayMessage p789]
[$e p247]
[$EmptyEnvironment p789]
[$describeOptions p789]

— defun describeSpad2Cmd —

(defun describeSpad2Cmd) (1)
(labels (fullname (arg)
  "Convert abbreviations to the full constructor name"
  (let ((abb (getdatabase arg 'abbreviation)))
    (if abb arg (getdatabase arg 'constructor)))
  (describeInternal (cdp internal?)
  (if internal?
(progn
  (unless (eq (getdatabase cdp 'constructor-kind) '|category|) (|dc| cdp))
  (showdatabase cdp))
  (mapcar #'(lambda (x) (if (stringp x) (cleanline x)))
    (flatten (car (getdatabase (fullname cdp) 'documentation))))))
(let ((|$e| |$EmptyEnvironment|) (opt (second l)))
  (declare (special |$e| |$EmptyEnvironment| $describeOptions))
  (if (and (consp l) (not (eq opt '?)))
    (describeInternal (first l) (second l))
    (|sayMessage|)
    (append
     '('" describe keyword arguments are"
     (mapcar #'(lambda (x) (format nil "-%a x)) $describeOptions)
     (format nil "-% or abbreviations thereof"))))))))

defun cleanline

— defun cleanline —

(defun cleanline (line)
  (labels
    (replaceInLine (thing other line)
      (do ((mark (search thing line) (search thing line)))
        ((null mark) line)
        (setq line
          (concatenate 'string (subseq line 0 mark) other
            (subseq line (+ mark (length thing))))))))
    (removeFromLine (thing line) (replaceInLine thing "" line))
    (removeKeyword (str line)
      (do ((mark (search str line) (search str line)))
        ((null mark) line)
        (let (left point mid right)
          (setq left (subseq line 0 mark))
          (setq point (search "}" line :start2 mark))
          (setq mid (subseq line (+ mark (length str)) point))
          (setq right (subseq line (+ point 1)))
          (setq line (concatenate 'string left mid right))))))
    (addSpaces (str line)
      (do ((mark (search str line) (search str line)) (cnt))
        ((null mark) line)
        (let (left point mid right)
          (setq left (subseq line 0 mark)))
...
(setq point (search "\}" line :start2 mark))
(setq mid (subseq line (+ mark (length str)) point))
(if (setq cnt (parse-integer mid :junk-allowed t))
 (setq mid (make-string cnt :initial-element #\ ))
 (setq mid ""))
(setq right (subseq line (+ point 1)))
(setq line (concatenate 'string left mid right])))

(splitAtNewline (line)
 (do ((mark (search "~\%" line) (search "~\%" line)) (lines))
 ((null mark)
 (push " " lines)
 (push line lines)
 (nreverse lines))
 (push (subseq line 0 mark) lines)
 (setq line (subseq line (+ mark 2))))))

(wrapOneLine (line margin result)
 (if (null line)
 (nreverse result)
 (if (< (length line) margin)
 (wrapOneLine nil margin (append (list line) result))
 (let (oneline spill aspace)
 (setq aspace (position #\space (subseq line 0 margin) :from-end t))
 (setq oneline (string-trim '(#\space) (subseq line 0 aspace)))
 (setq spill (string-trim '(#\space) (subseq line aspace)))
 (wrapOneLine spill margin (append (list oneline) result))))))

(reflowParagraph (line)
 (let (lst1)
 (setq lst1 (splitAtNewLine line))
 (dolist (x lst1)
 (mapcar #'(lambda(y) (format t "~a~\%" y))
 (wrapOneLine x 70 nil))))))

(setq line (removeFromLine "\}" line))
(setq line (replaceInLine "\\blankline" "~\%~\%" line))
(setq line (replaceInLine "\\br" "~\%" line))
(setq line (removeFromLine "\\" line))
(dolist (str '("spad" "spadtype" "spadop" "spadfun" "spadatt{"
 "axiom{" "axiomType" "spadignore{" "axiomFun{" "axiomUp{"
 "centerline{" "inputbitmap{" "axiomOp{" "spadgloss{")))
 (setq line (removeKeyword str line)))
(setq line (replaceInLine "{e.g.}" "e.g." line))
(dolist (str '("tab" "indented{"))
 (setq line (addSpaces str line)))
(reflowParagraph line))
defun flatten

— defun flatten 0 —

(defun flatten (x)
(labels (rec (x acc)
  (cond ((null x) acc)
        ((atom x) (cons x acc))
        (t (rec (car x) (rec (cdr x) acc))))))
(rec x nil))
25.18  )display Command

    display man page

 — display.help —

====================================================================
A.8.  )display
====================================================================

User Level Required:  interpreter

Command Syntax:

 - )display all
 - )display properties
 - )display properties all
 - )display properties [obj1 [obj2 ...] ]
 - )display value all
 - )display value [obj1 [obj2 ...] ]
 - )display mode all
 - )display mode [obj1 [obj2 ...] ]
 - )display names
 - )display operations opName

Command Description:

This command is used to display the contents of the workspace and signatures
of functions with a given name. (A signature gives the argument and return
types of a function.)

The command

)display names

lists the names of all user-defined objects in the workspace. This is useful
if you do not wish to see everything about the objects and need only be
reminded of their names.

The commands

)display all
)display properties
)display properties all

all do the same thing: show the values and types and declared modes of all
variables in the workspace. If you have defined functions, their signatures
and definitions will also be displayed.
To show all information about a particular variable or user functions, for example, something named \texttt{d}, issue

\texttt{)display properties d}

To just show the value (and the type) of \texttt{d}, issue

\texttt{)display value d}

To just show the declared mode of \texttt{d}, issue

\texttt{)display mode d}

All modemaps for a given operation may be displayed by using \texttt{)display} operations. A modemap is a collection of information about a particular reference to an operation. This includes the types of the arguments and the return value, the location of the implementation and any conditions on the types. The modemap may contain patterns. The following displays the modemaps for the operation \texttt{FromcomplexComplexCategory}:

\texttt{)d op complex}

Also See:

- \texttt{)clear}
- \texttt{)history}
- \texttt{)set}
- \texttt{)show}
- \texttt{)what}

---

\texttt{\textbf{defvar $displayOptions}}

The current value of \texttt{$displayOptions} is

- \texttt{initvars -}

\texttt{(defvar |$displayOptions| '(|abbreviations| |all| |macros| |modes| |names| |operations| |properties| |types| |values|))}

---

\texttt{\textsuperscript{6} “clear” (25.12 p 767) “history” (25.23 p 821) “set” (25.51 p 1013) “show” (25.52 p 1019) “what” (25.62 p 1091)}
defun display

This trivial function satisfies the standard pattern of making a user command match the name of the function which implements the command. That command immediately invokes a “Spad2Cmd” version. [displayspad2cmd p796]

— defun display —

(defun |display| (l)
  (displaySpad2Cmd l))

——

displaySpad2Cmd

We process the options to the command and call the appropriate display function. There are really only 4 display functions. All of the other options are just subcases.

There is a slight mismatch between the $displayOptions list of symbols and the options this command accepts so we have a cond branch to clean up the option variable. This allows for the options to be plural.

If we fall all the way thru we use the $displayOptions list to construct a list of strings for the sayMessage function and tell the user what options are available. [abbQuery p797]

[opOf p797]
[listConstructorAbbreviations p756]
[displayOperations p798]
[displayMacros p799]
[displayWorkspaceNames p726]
[displayProperties p733]
[selectOptionLC p751]
[sayMessage p797]
[$e p247]
[$EmptyEnvironment p795]
[$displayOptions p795]

— defun displaySpad2Cmd —

(defun displaySpad2Cmd (l)
  (let ((|$e| |$EmptyEnvironment|) (opt (car l)) (vl (cdr l)) option)
    (declare (special |$e| |$EmptyEnvironment| |$displayOptions|))
    (if (and (consp l) (not (eq opt '?)))
      (progn
        (setq option (|selectOptionLC| opt |$displayOptions| '|optionError|))
        (cond
          ((eq option '|all|)
            (setq l (list '|properties|))))
      )
  ))
(setq option '|properties|)
((or (eq option '|modes|) (eq option '|types|))
 (setq l (cons '|type| vl))
 (setq option '|type|))
((eq option '|values|)
 (setq l (cons '|value| vl))
 (setq option '|value|)))
(cond
 ((eq option '|abbreviations|)
  (if (null vl)
   (listConstructorAbbreviations)
   (dolist (v vl) (abbQuery (opOf v)))))
 ((eq option '|operations|) (displayOperations vl))
 ((eq option '|macros|) (displayMacros vl))
 ((eq option '|names|) (displayWorkspaceNames))
 (t (displayProperties option l)))))
(sayMessage
 (append
 '(" )display keyword arguments are")
 (mapcar #'(lambda (x) (format nil "\%a") $displayOptions)
 (format nil "\% or abbreviations thereof"))))

defun abbQuery

[getdatabase p1156]
[sayKeyedMsg p27]

— defun abbQuery —

(defun |abbQuery| (x)
 (let (abb)
  (cond
   ((setq abb (getdatabase x 'abbreviation))
    (sayKeyedMsg "\%1 abbreviates \%2 \%3"
     (list abb (getdatabase x 'constructorkind) x)))
   ((setq abb (getdatabase x 'constructor))
    (sayKeyedMsg "\%1 abbreviates \%2 \%3"
     (list x (getdatabase abb 'constructorkind) abb)))
   (t
    (sayKeyedMsg
     "\%1 is neither a constructor name nor a constructor abbreviation."
     (list x))))))
defun displayOperations

This function takes a list of operation names. If the list is null we query the user to see if they want all operations printed. Otherwise we print the information for the requested symbols.

[reportOpSymbol p798]
[yesanswer p798]
[sayKeyedMsg p27]

(defun displayOperations ()
  (if l
    (dolist (op l) (reportOpSymbol op))
    (if (yesanswer)
      (dolist (op (allOperations)) (reportOpSymbol op))
      (sayKeyedMsg
       (format nil
       "Since you did not respond with y or yes the list of operations will "
       "not be displayed.")
       nil))))

defun yesanswer

This is a trivial function to simplify the logic of displaySpad2Cmd. If the user didn’t supply an argument to the )display op command we ask if they wish to have all information about all Axiom operations displayed. If the answer is either Y or YES we return true else nil.

[string2id-n p1206]
[upcase p1206]
[queryUserKeyedMsg p27]

(defun yesanswer ()
  (member yesanswer
    (string2id-n
      (upcase
        (queryUserKeyedMsg
         (format nil
         "You have requested that all information about all Axiom operations ~
         (functions) be displayed. As there are several hundred operations, ~
         please confirm your request by typing y or yes and then pressing ~
         Enter :")
         nil)) i) '(y yes)))
defun displayMacros

(defun displayMacros (names)
  (let (imacs pmacs macros first)
    (setq imacs (getInterpMacroNames))
    (setq pmacs (getParserMacroNames))
    (if names
      (setq macros names)
      (setq macros (append imacs pmacs)))
    (setq macros (remdup macros))
    (cond
      ((null macros) (sayBrightly " There are no Axiom macros."))
      (t
       (setq first t)
       (do ((t0 macros (cdr t0)) (macro nil))
           ((or (atom t0) (progn (setq macro (car t0)) nil)) nil)
         (seq
          (exit
           (cond
            ((member macro pmacs)
             (cond
              (first (sayBrightly)
                (cons '|%l| (cons "User-defined macros:" nil)))
             (|displayParserMacro| macro))
            ((member macro imacs) '|iterate|)
            (t (sayBrightly)
             (cons " "
                 (cons macro
                   (cons " is not a known Axiom macro." nil)))))
             (setq first t)
             (do ((t1 macros (cdr t1)) (macro nil))
                 ((or (atom t1) (progn (setq macro (car t1)) nil)) nil)
                 (seq
                  (exit
                   (cond
                    ((member macro pmacs)
                     (cond
                      (first (sayBrightly)
                        (cons '|%l| (cons "User-defined macros:" nil)))
                     (|displayParserMacro| macro))
                    ((member macro imacs) '|iterate|)
                    (t (sayBrightly)
                      (cons " "
                        (cons macro
                          (cons " is not a known Axiom macro." nil))))))))))
  )
)
defun sayExample

This function expects 2 arguments, the documentation string and the name of the operation. It searches the documentation string for ++X lines. These lines are examples lines for functions. They look like ordinary ++ comments and fit into the ordinary comment blocks. So, for example, in the plot.spad.pamphlet file we find the following function signature:

\begin{verbatim}
plot: (F -> F,R) -> %
++ plot(f,a..b) plots the function \spad{f(x)}
++ on the interval \spad{[a,b]}.
++
++X fp:=(t:DFLOAT):DFLOAT +-> sin(t)
++X plot(fp,-1.0..1.0)$PLOT
\end{verbatim}

This function splits out and prints the lines that begin with ++X.

A minor complication of printing the examples is that the lines have been processed into internal compiler format. Thus the lines that read:

\begin{verbatim}
++X fp:=(t:DFLOAT):DFLOAT +-> sin(t)
++X plot(fp,-1.0..1.0)$PLOT
\end{verbatim}

are actually stored as one long line containing the example lines:

```
"{\textbackslash{\textbackslash indented}\{}{\textbackslash{\textbackslash spad}\{}f{\}},\{}{\textbackslash{\textbackslash spad}\{}a..\textbackslash{\textbackslash spad}\{}b\}{\}} \text{plots the function} \text{\textbackslash{\textbackslash spad}\{}f{\}}(x) \text{on the interval} \text{\textbackslash{\textbackslash spad}\}a..\textbackslash{\textbackslash spad}\{}b\}{\}}.
\text{\textbackslash{\textbackslash blankline}}
\text{\textbackslash{\textbackslash spad}\{}fp{\}}:=(t\textbackslash{\textbackslash spad}\{}DFLOAT\textbackslash{\textbackslash spad}\}+\rightarrow \text{\textbackslash{\textbackslash spad}\}sin\textbackslash{\textbackslash spad}\{(t\textbackslash{\textbackslash spad}\})
\text{\textbackslash{\textbackslash spad}\} fp\textbackslash{\textbackslash spad}\}(-1.0\textbackslash{\textbackslash spad}\}..\textbackslash{\textbackslash spad}\}1.0)\textbackslash{\textbackslash spad}\}PLOT"
```

So when we have an example line starting with ++X, it gets converted to the compiler to \spad{X}. So each example line is delimited by \spad{X}.

The compiler also removes the newlines so if there is a subsequent \spad{X} in the docstring then it implies multiple example lines and we loop over them, splitting them up at the delimiter.

If there is only one then we clean it up and print it.

\[\text{[cleanupLine p??]}\]
\[\text{[sayNewLine p??]}\]

—— defun sayExample ——

(defun sayExample (docstring)
  (let (line point)
    (when (setq point (search "spad{X}" docstring))
      (setq line (subseq docstring (+ point 8)))
      (do ((mark (search "spad{X}" line) (search "spad{X}" line)))
           ((null mark))
        (princ (cleanupLine (subseq line 0 mark)))
        (terpri)
        (setq line (subseq line (+ mark 8)))))
    (princ (cleanupLine line))
    (terpri)
    (terpri))))

defun cleanupLine

This function expects example lines in internal format that has been partially processed to remove the prefix. Thus we get lines that look like:

\[\text{fp:=(t:DFLOAT):DFLOAT +-> sin(\spad{t})}\]
\[\text{plot(\spad{fp},{}\spad{-1}.0..1.0)\$PLOT}\]

It removes all instances of {}, and \, and unwraps the \spad{} call, leaving only the argument.

We return lines that look like:

\[\text{fp:=(t:DFLOAT):DFLOAT +-> sin(t)}\]
\[\text{plot(fp,-1.0..1.0)$PLOT}\]

which is hopefully exactly what the user wrote.

The compiler inserts {} as a space so we remove it. We remove all of the \ characters. We remove all of the \spad{} delimiters which will occur around other \spad{} variables. Technically we should search recursively for the matching delimiter rather than the next
brace but the problem does not arise in practice.

-- defun cleanupLine 0 --

(defun cleanupLine (line)
  (do ((mark (search "\{" line) (search "\}" line)))
      ((null mark))
    (setq line
      (concatenate 'string (subseq line 0 mark) (subseq line (+ mark 2)))))
  (do ((mark (search "\\" line) (search "\\" line)))
      ((null mark))
    (setq line
      (concatenate 'string (subseq line 0 mark) (subseq line (+ mark 1)))))
  (do ((mark (search "spad{" line) (search "spad{" line)))
      ((null mark))
    (let (left point mid right)
      (setq left (subseq line 0 mark))
      (setq point (search "}" line :start2 mark))
      (setq mid (subseq line (+ mark 5) point))
      (setq right (subseq line (+ point 1)))
      (setq line (concatenate 'string left mid right))))
  line)
25.19 ）edit Command

edit man page

— edit.help —

====================================================================
A.9. ）edit
====================================================================

User Level Required: interpreter

Command Syntax:

- ）edit [filename]

Command Description:

This command is used to edit files. It works in conjunction with the ）read
and ）compile commands to remember the name of the file on which you are
working. By specifying the name fully, you can edit any file you wish. Thus

）edit /u/julius/matrix.input

will place you in an editor looking at the file /u/julius/matrix.input. By
default, the editor is vi, but if you have an EDITOR shell environment
variable defined, that editor will be used. When AXIOM is running under the X
Window System, it will try to open a separate xterm running your editor if it
thinks one is necessary. For example, under the Korn shell, if you issue

export EDITOR=emacs

then the emacs editor will be used by ）edit.

If you do not specify a file name, the last file you edited, read or compiled
will be used. If there is no ‘‘last file’’ you will be placed in the editor
editing an empty unnamed file.

It is possible to use the ）system command to edit a file directly. For
example,

）system emacs /etc/rc.tcpip

calls emacs to edit the file.

Also See:
\* ）system
\* ）compile
804  CHAPTER 25. SYSTEM COMMAND HANDLING

edit

7
defun edit

[editSpad2Cmd p804]

— defun edit —

(defun |edit| (l) (|editSpad2Cmd| l))

— defun editSpad2Cmd —

(defun |editSpad2Cmd| (l)
  (let (olddir filetypes ll rc)
    (declare (special /editfile))
    (setq l (cond ((null l) /editfile) (t (car l))))
    (setq l (|pathname| l))
    (setq olddir (|pathnameDirectory| l))
    (setq filetypes
      (cond
        (|pathnameType| l) (list (|pathnameType| l)))
        ((eq |$UserLevel| '|interpreter|) '("input" "INPUT" "spad" "SPAD"))
        ((eq |$UserLevel| '|compiler|) '("input" "INPUT" "spad" "SPAD"))
        (t '("input" "INPUT" "spad" "SPAD" "boot" "BOOT"
          "lisp" "LISP" "meta" "META"))))
    (setq ll 11)

7 “system” (25.56 p 1041) “read” (25.31 p 875)
defun Implement the )edit command

|strconc p?]| [namestring p1190] |pathname p1192] [obey p?]

— defun editFile —

(defun |editFile| (file)
  (cond
    ((member (intern "WIN32" (find-package 'keyword)) *features*)
      (obey (strconc "notepad " (|namestring| (|pathname| file))))
    (t
      (obey
        (strconc "$AXIOM/lib/SPADEDIT " (|namestring| (|pathname| file)))))))

— spadedit —

#!/bin/sh
# this script is invoked by the spad )edit command
# can be replaced by users favorite editor
# optional second argument should be character offset in file

thefile=$1
if [ ! -f $1 ] ; then

The SPADEDIT command

Axiom execute a shell script called SPADEDIT to open a file using the user’s chosen editor. That editor name is, by convention, in the EDITOR shell variable. If that variable is not set we default to the ‘vi’ editor.
thefile=$AXIOM/../../src/algebra/$1
else
  thefile=$1
fi

if [ $# = 2 ]; then
  START=`grep -n ^$2( $thefile | awk -F: '{print $1}'`
else
  START=1
fi

if [ ! "$EDITOR" ]; then
  EDITOR=vi
fi

if [ "$DISPLAY" ]; then
  if [ "$EDITOR" = "emacs" ]; then
    emacs +$START $thefile &
  elif [ "$EDITOR" = "vi" ]; then
    xterm -e vi +$START $thefile &
  else
    xterm -e $EDITOR $thefile &
  fi
else
  $EDITOR $thefile
fi

-----

defun updateSourceFiles
[pathname p1192]
[pathnameName p1191]
[pathnameType p1191]
[makeInputFilename p1129]
[member p1198]
[pathnameTypeId p1191]
[insert p??]
[$sourceFiles p??]

— defun updateSourceFiles —

(defun |updateSourceFiles| (arg)
  (declare (special |$sourceFiles|))
  (setq arg (|pathname| arg))
  (setq arg (|pathname| (list (|pathnameName| arg) (|pathnameType| arg) "*"))
  (when (and (makeInputFilename arg)

---
(member (pathnameTypeId arg) '(boot lisp meta))
(setq $sourceFiles (insert arg $sourceFiles)))
25.20  )fin Command

fin man page

--- fin.help ---

A.10.  )fin

User Level Required:  development

Command Syntax:

-  )fin

Command Description:

This command is used by AXIOM developers to leave the AXIOM system and return to the underlying Lisp system. To return to AXIOM, issue the "'(spad)" function call to Lisp.

Also See:

o  )pquit
o  )quit

---

\[\text{defun Exit from the interpreter to lisp}\]

\[\text{[spad-reader p??]}
\[\text{[eof p??]}

--- defun fin 0 ---

(defun fin ()
  (setq *eof* t)
  (throw 'spad_reader nil))

---

\(^8\) "pquit" (25.29 p 870) "quit" (25.30 p 872)
This command is in the list of $noParseCommands 25.1 which means that its arguments are passed verbatim. This will eventually result in a call to the function handleNoParseCommands 25.2
25.21 \)help Command

\)help man page

— help.help —

====================================================================
A.12. \)help
====================================================================

User Level Required: interpreter

Command Syntax:
- \)help
- \)help commandName
- \)help syntax

Command Description:
This command displays help information about system commands. If you issue \)help
then this very text will be shown. You can also give the name or abbreviation of a system command to display information about it. For example,

\)help clear

will display the description of the \)clear system command.

The command \)help syntax

will give further information about the Axiom language syntax.

All this material is available in the AXIOM User Guide and in HyperDoc. In HyperDoc, choose the Commands item from the Reference menu.

====================================================================
A.1. Introduction
====================================================================

System commands are used to perform AXIOM environment management. Among the commands are those that display what has been defined or computed, set up multiple logical AXIOM environments (frames), clear definitions, read files
of expressions and commands, show what functions are available, and terminate AXIOM.

Some commands are restricted: the commands

)set userlevel interpreter
)set userlevel compiler
)set userlevel development

set the user-access level to the three possible choices. All commands are available at development level and the fewest are available at interpreter level. The default user-level is interpreter. In addition to the )set command (discussed in description of command )set ) you can use the HyperDoc settings facility to change the user-level. Click on [Settings] here to immediately go to the settings facility.

Each command listing begins with one or more syntax pattern descriptions plus examples of related commands. The syntax descriptions are intended to be easy to read and do not necessarily represent the most compact way of specifying all possible arguments and options; the descriptions may occasionally be redundant.

All system commands begin with a right parenthesis which should be in the first available column of the input line (that is, immediately after the input prompt, if any). System commands may be issued directly to AXIOM or be included in .input files.

A system command argument is a word that directly follows the command name and is not followed or preceded by a right parenthesis. A system command option follows the system command and is directly preceded by a right parenthesis. Options may have arguments: they directly follow the option. This example may make it easier to remember what is an option and what is an argument:

)syscmd arg1 arg2 )opt1 opt1arg1 opt1arg2 )opt2 opt2arg1 ...

In the system command descriptions, optional arguments and options are enclosed in brackets ('[' and ']'). If an argument or option name is in italics, it is meant to be a variable and must have some actual value substituted for it when the system command call is made. For example, the syntax pattern description

)read fileName []quietly]

would imply that you must provide an actual file name for fileName but need not use the )quietly option. Thus

)read matrix.input

is a valid instance of the above pattern.
CHAPTER 25. SYSTEM COMMAND HANDLING

System command names and options may be abbreviated and may be in upper or lower case. The case of actual arguments may be significant, depending on the particular situation (such as in file names). System command names and options may be abbreviated to the minimum number of starting letters so that the name or option is unique. Thus

)s Integer

is not a valid abbreviation for the )set command, because both )set and )show begin with the letter ‘s’. Typically, two or three letters are sufficient for disambiguating names. In our descriptions of the commands, we have used no abbreviations for either command names or options.

In some syntax descriptions we use a vertical line ‘|’ to indicate that you must specify one of the listed choices. For example, in

)set output fortran on | off

only on and off are acceptable words for following boot. We also sometimes use ‘...’ to indicate that additional arguments or options of the listed form are allowed. Finally, in the syntax descriptions we may also list the syntax of related commands.

Other help topics

Available help topics are:

<table>
<thead>
<tr>
<th>abbreviations</th>
<th>assignment</th>
<th>blocks</th>
<th>browse</th>
<th>boot</th>
<th>cd</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear</td>
<td>clef</td>
<td>close</td>
<td>collection</td>
<td>compile</td>
<td>describe</td>
</tr>
<tr>
<td>display</td>
<td>edit</td>
<td>fin</td>
<td>for</td>
<td>frame</td>
<td>help</td>
</tr>
<tr>
<td>history</td>
<td>if</td>
<td>iterate</td>
<td>leave</td>
<td>library</td>
<td>lisp</td>
</tr>
<tr>
<td>load</td>
<td>ltrace</td>
<td>parallel</td>
<td>pquit</td>
<td>quit</td>
<td>read</td>
</tr>
<tr>
<td>repeat</td>
<td>savesystem</td>
<td>set</td>
<td>show</td>
<td>spool</td>
<td>suchthat</td>
</tr>
<tr>
<td>synonym</td>
<td>system</td>
<td>syntax</td>
<td>trace</td>
<td>undo</td>
<td>what</td>
</tr>
<tr>
<td>while</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Available algebra help topics are:

The top level help command

[helpSpad2Cmd p813]

--- defun help ---
(defun |help| (l)
  "The top level help command"
  (|helpSpad2Cmd| l))

---

The top level help command handler

[newHelpSpad2Cmd p813]
[sayKeyedMsg p27]

---

— defun helpSpad2Cmd —

(defun |helpSpad2Cmd| (args)
  "The top level help command handler"
  (unless (|newHelpSpad2Cmd| args)
    (|sayKeyedMsg|
     (format nil
      "If the system command or synonym %1 exists, help information is not ~
       available for it. Issue )what commands or )what synonyms to ~
       determine is %1 is a valid name."
      (cons args nil))))

---

defun newHelpSpad2Cmd

[makeInputFilename p1129]
[obey p??]
[concat p1197]
[namestring p1190]
[make-instream p1127]
[say p??]
[abbreviation? p??]
[poundsign p??]
[sayKeyedMsg p27]
[pname p1195]
[selectOptionLC p751]
[$syscommands p716]
[$useFullScreenHelp p936]

---

— defun newHelpSpad2Cmd —

(defun |newHelpSpad2Cmd| (args)
(let (sarg arg narg helpfile filestream line unabbrev)
  (declare (special $syscommands |$useFullScreenHelp|))
  (when (null args) (setq args (list '?)))
  (if (> (|#| args) 1)
      (|sayKeyedMsg| "The )help system command supports at most one argument." nil)
      (progn
        (setq sarg (pname (car args)))
        (cond
          ((string= sarg "?") (setq args (list '|help|)))
          ((string= sarg "%") (setq args (list '|history|)))
          ((string= sarg "%%") (setq args (list '|history|)))
          (t nil))
        (setq arg (|selectOptionLC| (car args) $syscommands nil))
        (cond ((null arg) (setq arg (car args))))
        (setq narg (pname arg))
        ; expand abbreviations to full constructor names
        (when
          (setq unabbrev (|abbreviation?| (intern narg)))
          (setq narg (symbol-name unabbrev))))
      (cond
        ; if the help file does not exist, exit
        ((null (setq helpfile (makeInputFilename (list narg "help")))) nil)
      ; if we expect to use full screen help, call SPADEDIT
      ($useFullScreenHelp|
        (obey (concat "$AXIOM/lib/SPADEDIT " (|namestring| helpfile))) t)
      ; otherwise dump the help file to the console
      (t
        (setq filestream (make-instream helpfile))
        (do ((line (|read-line| filestream nil) (|read-line| filestream nil)))
          (null line) (shut filestream))
        (say line)))))))
25.22  )HISTORY COMMAND

25.22  )history Command

history man page

— history.help —

====================================================================
A.13.  )history
====================================================================

User Level Required:  interpreter

Command Syntax:

-  )history on
-  )history off
-  )history write historyInputFileName
-  )history show [n] [both]
-  )history save savedHistoryName
-  )history restore [savedHistoryName]
-  )history reset
-  )history change n
-  )history memory
-  )history file
-  %
-  %%(n)
-  )set history on | off

Command Description:

The history facility within AXIOM allows you to restore your environment to
that of another session and recall previous computational results. Additional
commands allow you to review previous input lines and to create an .input
file of the lines typed to AXIOM.

AXIOM saves your input and output if the history facility is turned on (which
is the default). This information is saved if either of

)set history on
)history on

has been issued. Issuing either

)set history off
)history off

will discontinue the recording of information.
Whether the facility is disabled or not, the value of % in AXIOM always refers to the result of the last computation. If you have not yet entered anything, % evaluates to an object of type Variable('%). The function ‰ may be used to refer to other previous results if the history facility is enabled. In that case, ‰(n) is the output from step n if n > 0. If n < 0, the step is computed relative to the current step. Thus ‰(-1) is also the previous step, ‰(-2), is the step before that, and so on. If an invalid step number is given, AXIOM will signal an error.

The environment information can either be saved in a file or entirely in memory (the default). Each frame has its own history database. When it is kept in a file, some of it may also be kept in memory for efficiency. When the information is saved in a file, the name of the file is of the form FRAME.axh where ‘FRAME’ is the name of the current frame. The history file is placed in the current working directory (see description of command cd). Note that these history database files are not text files (in fact, they are directories themselves), and so are not in human-readable format.

The options to the )history command are as follows:

)change n
   will set the number of steps that are saved in memory to n. This option only has effect when the history data is maintained in a file. If you have issued )history )memory (or not changed the default) there is no need to use )history )change.

)on
   will start the recording of information. If the workspace is not empty, you will be asked to confirm this request. If you do so, the workspace will be cleared and history data will begin being saved. You can also turn the facility on by issuing )set history on.

)off
   will stop the recording of information. The )history )show command will not work after issuing this command. Note that this command may be issued to save time, as there is some performance penalty paid for saving the environment data. You can also turn the facility off by issuing )set history off.

)file
   indicates that history data should be saved in an external file on disk.

)memory
   indicates that all history data should be kept in memory rather than saved in a file. Note that if you are computing with very large objects it may not be practical to keep this data in memory.

)reset
   will flush the internal list of the most recent workspace calculations so
that the data structures may be garbage collected by the underlying Lisp system. Like )history )change, this option only has real effect when history data is being saved in a file.

)restore [savedHistoryName]
completely clears the environment and restores it to a saved session, if possible. The )save option below allows you to save a session to a file with a given name. If you had issued )history )save jacobi the command )history )restore jacobi would clear the current workspace and load the contents of the named saved session. If no saved session name is specified, the system looks for a file called last.axh.

)save savedHistoryName
is used to save a snapshot of the environment in a file. This file is placed in the current working directory (see description of command )cd ). Use )history )restore to restore the environment to the state preserved in the file. This option also creates an input file containing all the lines of input since you created the workspace frame (for example, by starting your AXIOM session) or last did a )clear all or )clear completely.

)show [n] [both]
can show previous input lines and output results. )show will display up to twenty of the last input lines (fewer if you haven’t typed in twenty lines). )show n will display up to n of the last input lines. )show both will display up to five of the last input lines and output results. )show n both will display up to n of the last input lines and output results.

)write historyInputFile
creates an .input file with the input lines typed since the start of the session/frame or the last )clear all or )clear completely. If historyInputFileName does not contain a period (".") in the filename, .input is appended to it. For example, )history )write chaos and )history )write chaos.input both write the input lines to a file called chaos.input in your current working directory. If you issued one or more )undo commands, )history )write eliminates all input lines backtracked over as a result of )undo. You can edit this file and then use )read to have AXIOM process the contents.

Also See:
 o )frame
 o )read
 o )set
 o )undo

---

9 “frame” (2.5 p 10) “read” (25.31 p 875) “set” (25.51 p 1013) “undo” (25.60 p 1072)
CHAPTER 25. SYSTEM COMMAND HANDLING

History recording is done in two different ways:

- all changes in variable bindings (i.e. previous values) are written to \$HistList, which is a circular list
- all new bindings (including the binding to %) are written to a file called histFileName()

one older session is accessible via the file \$oldHistFileName()

25.23 Initialized history variables

The following global variables are used:

\$HistList, \$HistListLen and \$HistListAct which is the actual number of “undoable” steps

\$HistRecord collects the input line, all variable bindings and the output of a step, before it is written to the file histFileName().

\$HiFiAccess is a flag, which is reset by )history )off

The result of step n can be accessed by \%n, which is translated into a call of fetchOutput(n).

The updateHist is called after every interpreter step. The putHist function records all changes in the environment to \$HistList and \$HistRecord.

defvar \$oldHistoryFileName

— initvars —

(defvar \$oldHistoryFileName| 'last| "vm/370 filename name component")

——

defvar \$historyFileType

— initvars —

(defvar \$historyFileType| 'axh| "vm/370 filename type component")

——
defvar $historyDirectory

    — initvars —

    (defvar $historyDirectory 'A "vm/370 filename disk component")

-----

defvar $useInternalHistoryTable

    — initvars —

    (defvar $useInternalHistoryTable t "t means keep history in core")

-----

defun makeHistFileName

    [makePathname p1193]

    — defun makeHistFileName —

    (defun makeHistFileName (fname)
        (makePathname fname |$historyFileType| |$historyDirectory|))

-----

defun oldHistFileName

    [makeHistFileName p819]
    [$OldHistoryFileName p818]

    — defun oldHistFileName —

    (defun oldHistFileName ()
        (reject (special |$oldHistoryFileName|))
        (makeHistFileName |$oldHistoryFileName|))
defun histFileName

[makeHistFileName p819]
[$interpreterFrameName p23]

— defun histFileName —

(defun |histFileName| ()
(declare (special |$interpreterFrameName|))
(|makeHistFileName| |$interpreterFrameName|))

—

defun histInputFileName

[makePathname p1193]
[$interpreterFrameName p23]
[$historyDirectory p819]

— defun histInputFileName —

(defun |histInputFileName| (fn)
(declare (special |$interpreterFrameName| |$historyDirectory|))
(if (null fn)
  (|makePathname| |$interpreterFrameName| 'input |$historyDirectory|)
  (|makePathname| fn 'input |$historyDirectory|)))

—

defun initHist

[initHistList p821]
[oldHistFileName p819]
[histFileName p820]
[histFileErase p860]
[makeInputFilename p1129]
[$replace p??]
[$useInternalHistoryTable p819]
[$HiFiAccess p937]

— defun initHist —

(defun |initHist| ()
(let (oldFile newFile)
(declare (special |$useInternalHistoryTable| |$HiFiAccess|))
(if |$useInternalHistoryTable|
   (|initHistList|)
   (progn
      (setq oldFile (|oldHistFileName|))
      (setq newFile (|histFileName|))
      (|histFileErase| oldFile)
      (when (makeInputFilename newFile) (replaceFile oldFile newFile))
      (setq |$HiFiAccess| t)
      (|initHistList|))))

defun initHistList

|$HistListLen p32|
|$HistList p31|
|$HistListAct p32|
|$HistRecord p32|

— defun initHistList —

(defun |initHistList| ()
  (let (li)
    (declare (special |$HistListLen| |$HistList| |$HistListAct| |$HistRecord|))
    (setq |$HistListLen| 20)
    (setq |$HistList| (list nil))
    (setq li |$HistList|)
    (do ((i 1 (1+ i)))
        (> i |$HistListLen|) nil)
        (setq li (cons nil li)))
    (rplacd |$HistList| li)
    (setq |$HistListAct| 0)
    (setq |$HistRecord| nil)))

————

The top level history command

[sayKeyedMsg p27]
[historySpad2Cmd p822]
[$options p??]

— defun history —
(defun |history| (l)
  "The top level history command"
  (declare (special |$options|))
  (if (or l (null |$options|))
    (|sayKeyedMsg|
     (format nil
      "You have not used the correct syntax for the history command. ~
       Issue )help history for more information."
     nil)
    (|historySpad2Cmd|)))

---

The top level history command handler

[selectOptionLC p751]
[member p1198]
[sayKeyedMsg p27]
[initHistList p821]
[upcase p1206]
[queryUserKeyedMsg p??]
[string2id-n p??]
[histFileErase p860]
[histFileName p820]
[clearSpad2Cmd p767]
[disableHist p845]
[setHistoryCore p757]
[resetInCoreHist p830]
[saveHistory p837]
[showHistory p824]
[changeHistListLen p830]
[restoreHistory p839]
[writeInputLines p828]
[seq p??]
[exit p??]
[$options p??]
[$HiFiAccess p937]
[$IOindex p24]

--- defun historySpad2Cmd ---

(defun |historySpad2Cmd| ()
  "The top level history command handler"
  (let (histOptions opts opt optargs x)
    (declare (special |$options| |$HiFiAccess| |$IOindex|))
    (setq histOptions
'(|on| |off| |yes| |no| |change| |reset| |restore| |write| |save| |show| |file| |memory|)

(setq opts
  (prog (tmp1)
    (setq tmp1 nil)
    (return
      (do ((tmp2 |$options| (cdr tmp2)) (tmp3 nil))
        ((or (atom tmp2)
            (progn
              (setq tmp3 (car tmp2))
              nil)
            (progn
              (progn
                (setq opt (car tmp3))
                (setq optargs (cdr tmp3))
                tmp3)
              nil))
      (nreverse0 tmp1))
    (setq tmp1
      (cons
        (cons
          ((selectOptionLC opt histOptions '|optionError|)
           optargs)
          tmp1))))
  (do ((tmp4 opts (cdr tmp4)) (tmp5 nil))
      ((or (atom tmp4)
           (progn
             (setq tmp5 (car tmp4))
             nil)
           (progn
             (progn
               (setq opt (car tmp5))
               (setq optargs (cdr tmp5))
               tmp5)
             nil))
      nil)
    (seq
     (exit
      (cond
        ((|member| opt '(|on| |yes|))
         (cond
          (|$HiFiAccess|
           (|sayKeyedMsg| "The history facility is already on." nil))
          ((eql |$IoIndex| 1)
           (setq |$HiFiAccess| t)
           (|initHistList|)
           (|sayKeyedMsg| "The history facility is now on." nil))
          (t
           (setq x ; really want to turn history on?
             (upcase
               (|selectOptionLC| opt histOptions '|optionError|)
               optargs)
             tmp4)))))

(defun |selectOptionLC| (opt histOptions |optionError|)
  (cond
    ((|member| opt '(|on| |yes|))
     (prog
      (setq tmp (car optargs))
      (return nil))
    ((|member| opt '(|off| |reset| |restore| |write| |save| |show| |file| |memory|))
      (setq tmp (car optargs))
      (return nil))
    ((eql |$IoIndex| 1)
      (setq |$HiFiAccess| t)
      (|initHistList|)
      (|sayKeyedMsg| "The history facility is now on." nil))
    (t
     (return nil))
    (nil
     (return nil)))

(defun |selectOptionLC| (opt histOptions |optionError|)
  (cond
    ((|member| opt '(|on| |yes|))
     (prog
      (setq tmp (car optargs))
      (return nil))
    ((|member| opt '(|off| |reset| |restore| |write| |save| |show| |file| |memory|))
      (setq tmp (car optargs))
      (return nil))
    ((eql |$IoIndex| 1)
      (setq |$HiFiAccess| t)
      (|initHistList|)
      (|sayKeyedMsg| "The history facility is now on." nil))
    (t
     (return nil))
    (nil
     (return nil)))
Turning on the history facility will clear the contents of the workspace. Please enter y or yes if you really want to do this:

```
(queryUserKeyedMsg
  (format nil "Turning on the history facility will clear the contents of the workspace. Please enter y or yes if you really want to do this:"
  nil)))
```

```
(cond
  ((member (string2id-n x 1) '(Y YES))
   (histFileErase (histFileName))
   (setq $HiFiAccess t)
   (setq $options nil)
   (clearSpad2Cmd '([all]))
   (sayKeyedMsg "The history facility is now on." nil)
   (initHistList))
  (t
   (sayKeyedMsg "The history facility is still off." nil))))
```

```
((member opt '(off |no|))
  (cond
   (null $HiFiAccess)
   (sayKeyedMsg "The history facility is already off." nil))
   (t
    (setq $HiFiAccess nil)
    (disableHist)
    (sayKeyedMsg "The history facility is now off." nil))))
```

```
((eq opt 'file)  (setHistoryCore nil))
((eq opt 'memory)  (setHistoryCore t))
((eq opt 'reset)  (resetInCoreHist))
((eq opt 'save)  (saveHistory optargs))
((eq opt 'show)  (showHistory optargs))
((eq opt 'change)  (changeHistListLen (car optargs)))
((eq opt 'restore)  (restoreHistory optargs))
((eq opt 'write)  (writeInputLines optargs 1))))
```

```
'done)
```

defun showHistory

```lisp
[sayKeyedMsg p27]
[selectOptionLC p751]
[sayMSG p29]
[concat p1197]
[bright p??]
[showInOut p842]
[setIOindex p841]
[showInput p841]
[$printTimeSum p??]
```
25.23. INITIALIZED HISTORY VARIABLES

[Example code]

(defun showHistory (arg)
  (let ((printTimeSum evalTimePrint maxi mini arg2 arg1
        nset n showInputOrBoth)
        (declare (special printTimeSum evalTimePrint HiFiAccess)))
    (setq evalTimePrint 0)
    (setq printTimeSum 0)
    (cond
      ((null HiFiAccess)
       (sayKeyedMsg
        (format nil
                  "The history facility command %1 cannot be performed because the "
                  "history facility is not on.")
                  (list 'show)))
      (t
       (setq showInputOrBoth 'input))
    (setq n 20)
    (when arg
      (setq arg1 (car arg))
      (when (integerp arg1)
        (setq n arg1)
        (setq nset t)
        (cond
         ((ifcdr arg) (setq arg1 (cadr arg)))
         (t (setq arg1 nil))))
    (when arg1
      (setq arg2 (selectOptionLC arg1 '(input both) nil))
    (cond
      (arg2
       (cond
        ((and (eq (setq showInputOrBoth arg2) 'both)
            (null nset))
         (setq n 5)))))
    (t
     (sayMsg
      (concat " " (bright arg1) " is an invalid argument."))))}
    (cond ((not (< n IDindex)) (setq n (- IDindex 1)))
      (setq mini (- IDindex n))
      (setq maxi (- IDindex 1))
    (cond
      ((eq showInputOrBoth 'both)
       (unwind-protect
        (showInOut mini maxi)
        (setq IDindex (+ maxi 1))))
      (t (showInput mini maxi))))
defun setHistoryCore

We case on the inCore argument value

If history is already on and is kept in the same location as requested (file or memory) then complain.

If history is not in use then start using the file or memory as requested. This is done by simply setting the $useInternalHistoryTable to the requested value, where T means use memory and NIL means use a file. We tell the user.

If history should be in memory, that is inCore is not NIL, and the history file already contains information we read the information from the file, store it in memory, and erase the history file. We modify $useInternalHistoryTable to T to indicate that we’re maintaining the history in memory and tell the user.

Otherwise history must be on and in memory. We erase any old history file and then write the in-memory history to a new file

— defun setHistoryCore —

(defun |setHistoryCore| (inCore)
  (let (l vec str n rec)
    (declare (special |$useInternalHistoryTable| |$internalHistoryTable|
                |$HiFiAccess| |$IOindex|))
    (cond
      ((boot-equal inCore |$useInternalHistoryTable|)
       (if inCore
           (|sayKeyedMsg|
            (format nil
                    "History information is already being maintained in memory (and ~
                     not in an external file.").")
            nil))
       nil))
(sayKeyedMsg)
(format nil
  "History information is already being maintained in an external ~
  file (and not in memory.).")
nil))) ; file history already in use
((null $HiFiAccess))
(setq $useInternalHistoryTable inCore)
(if inCore
  (sayKeyedMsg)
  (format nil
    "When the history facility is active, history information will be ~
    maintained in memory (and not in an external file.).")
  nil)
(sayKeyedMsg)
(format nil
  "When the history facility is active, history information will be ~
  maintained in a file (and not in an internal table.).")
nil)))
inCore
(setq $internalHistoryTable nil)
(cond
  ((not (eql $IOindex 0))
   (setq l (length (rkeyids (histFileName))))
   (do ((i 1 (1+ i)))
       ((> i l) nil)
       (setq vec (unwind-protect (readHiFi i) (disableHist)))
       (setq $internalHistoryTable
         (cons (cons i vec) $internalHistoryTable)))
   (histFileErase (histFileName)))
   (setq $useInternalHistoryTable t)
  (sayKeyedMsg)
  (format nil
    "When the history facility is active, history information will be ~
    maintained in memory (and not in an external file.).")
  nil))
(t
(setq $HiFiAccess nil)
(histFileErase (histFileName))
(setq str
  (rdefiostream
    (cons
      '(mode . output)
      (cons
        (cons 'file (histFileName)
          nil)))))
  (do ((tmp0 (reverse $internalHistoryTable) (cdr tmp0))
       (tmp1 nil)
       ((or (atom tmp0)
            (progn
              (setq tmp1 (car tmp0))))
       nil) i nil) (setq l (length (rkeyids (histFileName))))
       (do ((i 1 (1+ i)))
           ((> i l) nil)
           (setq vec (unwind-protect (readHiFi i) (disableHist)))
           (setq $internalHistoryTable
             (cons (cons i vec) $internalHistoryTable)))
           (histFileErase (histFileName)))
           (setq $useInternalHistoryTable t)
           (sayKeyedMsg)
           (format nil
             "When the history facility is active, history information will be ~
             maintained in memory (and not in an external file.).")
           nil))
(t
(setq $HiFiAccess nil)
(histFileErase (histFileName))
(setq str
  (rdefiostream
    (cons
      '(mode . output)
      (cons
        (cons 'file (histFileName)
          nil))))
  (do ((tmp0 (reverse $internalHistoryTable) (cdr tmp0))
       (tmp1 nil)
       ((or (atom tmp0)
            (progn
              (setq tmp1 (car tmp0))))
       nil) i nil) (setq l (length (rkeyids (histFileName))))
       (do ((i 1 (1+ i)))
           ((> i l) nil)
           (setq vec (unwind-protect (readHiFi i) (disableHist)))
           (setq $internalHistoryTable
             (cons (cons i vec) $internalHistoryTable)))
           (histFileErase (histFileName)))
           (setq $useInternalHistoryTable t)
           (sayKeyedMsg)
           (format nil
             "When the history facility is active, history information will be ~
             maintained in memory (and not in an external file.).")
           nil))
CHAPTER 25. SYSTEM COMMAND HANDLING

```
-nil)
(progn
  (progn
    (setq n (car tmp1))
    (setq rec (cdr tmp1))
    tmp1)
  nil))
  nil))
(spadrwrite (|object2Identifier| n) rec str))
(rshut str)
(setq |$HiFiAccess| t)
(setq |$internalHistoryTable| nil)
(setq |$useInternalHistoryTable| nil)
(|sayKeyedMsg|
  (format nil
    "When the history facility is active, history information will be ~
     maintained in a file (and not in an internal table)."
    nil)))

---

defvar $underbar

Also used in the output routines.

— initvars —

(defvar underbar "\_")

---

defun writeInputLines

[sayKeyedMsg p27]
[throwKeyedMsg p??]
[size p1196]
[concat p1197]
[substring p256]
[readHiFi p843]
[histInputFileName p820]
[histFileErase p860]
[defiostream p1128]
[namestring p1190]
[shut p1128]
[underbar p828]
— defun writeInputLines —

(defun writeInputLines (fn initial)
  (let ((maxn breakChars vecl k svec done n lineList file inp)
    (declare (special underbar $HiFiAccess $IOindex))
    (cond
      ((null $HiFiAccess)
        (sayKeyedMsg
          (format nil
            "The history facility is not on, so the .input file containing your ~
             user input cannot be created."
          )
        )
      )
      ((null fn)
        (throwKeyedMsg
          "You must specify a file name to the history write command" nil)
      )
      (t
        (setq maxn 72)
        (setq breakChars (cons '|' | (cons '+' nil)))
        (do ((tmp0 (- $IOindex 1)
              (i initial (+ i 1)))
             ((> i tmp0) nil)
            (setq vecl (car (readHiFi i)))
            (when (stringp vecl) (setq vecl (cons vecl nil)))
            (dolist (vec vecl)
              (setq n (size vec))
              (do ()
                ((null (> n maxn)) nil)
                (setq done nil)
                (do ((j 1 (1+ j))
                     ((or (> j maxn) (null (null done))) nil)
                    (setq k (- (1+ maxn) j))
                    (when (member (elt vec k) breakChars)
                      (setq svec (concat (substring vec 0 (1+ k)) underbar))
                      (setq lineList (cons svec lineList))
                      (setq done t)
                      (setq vec (substring vec (1+ k) nil))
                      (setq n (size vec)))
                )
                (when done (setq n 0))
                (setq lineList (cons vec lineList))))
        (setq file (histInputFileName fn))
        (histFileErase file)
        (setq inp
          (defiostream
            (cons
              '(mode . output)
              (cons (cons 'file file) nil)) 255 0))
        (dolist (x (removeUndolines (nreverse lineList))))
  )
)
(write-line x inp))
(cond
  ((not (eq fn '|redo|))
   (|sayKeyedMsg| "Edit %1 to see the saved input lines."
     (list (|namestring| file)))))
  (shut inp)
  nil)))

defun resetInCoreHist

[|HistListAct| p32]
[|HistListLen| p32]
[|HistList| p31]

— defun resetInCoreHist —

(defun |resetInCoreHist| ()
  (declare (special |HistListAct| |HistListLen| |HistList|))
  (setq |HistListAct| 0)
  (do ((i 1 (1+ i)))
      ((> i |HistListLen|) nil)
    (setq |HistList| (cdr |HistList|))
    (rplaca |HistList| nil)))

defun changeHistListLen

[|sayKeyedMsg| p27]
[|HistListLen| p32]
[|HistList| p31]
[|HistListAct| p32]

— defun changeHistListLen —

(defun |changeHistListLen| (n)
  (let (dif 1)
    (declare (special |HistListLen| |HistList| |HistListAct|))
    (if (null (integerp n))
      (|sayKeyedMsg|
        (format nil
          "The argument n for )history )change n must be a nonnegative integer ~
          and your argument, %1 , is not one.")
        nil)))
(list n)) ; only positive integers
(progn
  (setq dif (- n |$HistListLen|))
  (setq |$HistListLen| n)
  (setq l (cdr |$HistList|))
  (cond
    ((> dif 0)
     (do ((i 1 (1+ i)))
         ((> i dif) nil)
         (setq l (cons nil l)))))
  ((minusp dif)
   (do ((tmp0 (- dif))
        (i 1 (1+ i)))
       ((> i tmp0) nil)
       (setq l (cdr l)))
   (cond
    ((> |$HistListAct| n) (setq |$HistListAct| n))
    (t nil)))
  (rplacd |$HistList| l)
  '|done|)))

(defun updateHist
  (declare (special |$IOindex| |$HiFiAccess| |$HistRecord| |$mkTestInputStack|
                 |$currentLine|))
  (when |$IOindex|
    (startTimingProcess '|history|)
    (updateInCoreHist)
  (when |$HiFiAccess|
    (unwind-protect (writeHiFi) (disableHist))
    (updateCurrentInterpreterFrame)
    (stopTimingProcess)
    ($IOindex p24]
    [$HiFiAccess p937]
    [$HistRecord p32]
    [$mkTestInputStack p932]
    [$currentLine p932]

---
defun updateHist ---

(defun updateHist ()
  (declare (special |$IOindex| |$HiFiAccess| |$HistRecord| |$mkTestInputStack|
                   |$currentLine|))
  (when |$IOindex|
    (|startTimingProcess| '|history|)
    (|updateInCoreHist|)
  (when |$HiFiAccess|
    (unwind-protect (writeHiFi) (disableHist))
    (stopTimingProcess)
(setq "$HistRecord| nil))
(incf "$IOindex|)
(updateCurrentInterpreterFrame|)
(setq "$mkTestInputStack| nil)
(setq "$currentLine| nil)
(stopTimingProcess| 'history|))

defun updateInCoreHist

([^HistList p31])
([^HistListLen p32])
([^HistListAct p32])

— defun updateInCoreHist —

(defun |updateInCoreHist| ()
(declare (special "$HistList| "$HistListLen| "$HistListAct|)))
(setq "$HistList| (cdr "$HistList|))
(rplaca "$HistList| nil)
(when (> "$HistListLen| "$HistListAct|)
 (setq "$HistListAct| (1+ "$HistListAct|))))

defun putHist

[^recordOldValue p833]
[^get p??]
[^recordNewValue p833]
[^putIntSymTab p??]
[^HiFiAccess p937]

— defun putHist —

(defun |putHist| (x prop val e)
(declare (special "$HiFiAccess|)))
(when (null (eq x '|)) (|recordOldValue| x prop (get x prop e)))
(when ($HiFiAccess| (|recordNewValue| x prop val))
 (|putIntSymTab| x prop val e))
25.23. INITIALIZED HISTORY VARIABLES

defun recordNewValue

[setq p1200]
[$HistRecord p32]

— defun recordNewValue —

(defun recordNewValue (x prop val)
 (startTimingProcess 'history)
 (recordNewValue0 x prop val)
 (stopTimingProcess 'history))

———

defun recordNewValue0

[setq p1200]
[$HistRecord p32]

— defun recordNewValue0 —

(defun recordNewValue0 (x prop val)
 (let (p1 p2 p)
  (declare (special $HistRecord))
  (if (setq p1 (assq x $HistRecord))
   (if (setq p2 (assq prop (cdr p1)))
    (rplacd p2 val)
    (rplacd p1 (cons (cons prop val) (cdr p1))))
   (progn
    (setq p (cons x (list (cons prop val))))
    (setq $HistRecord (cons p $HistRecord))))))

———

defun recordOldValue

[setq p1200]
[$HistRecord p32]

— defun recordOldValue —
(defun |recordOldValue| (x prop val)
  ([startTimingProcess| '|history|])
  ([recordOldValue0] x prop val)
  ([stopTimingProcess| '|history|]))

---

defun recordOldValue0

[$HistList p31]

— defun recordOldValue0 —

(defun |recordOldValue0| (x prop val)
  (let (p1 p)
    (declare (special |$HistList|))
    (when (setq p1 (assq x (car |$HistList|)))
      (when (null (assq prop (cdr p1)))
        (rplacd p1 (cons (cons prop val) (cdr p1))))
    (setq p (cons x (list (cons prop val))))
    (rplaca |$HistList| (cons p (car |$HistList|))))

---

defun undoInCore

[undoChanges p835]
[readHiFi p843]
[disableHist p845]
[assq p1200]
[sayKeyedMsg p27]
[putHist p832]
[updateHist p831]
[$HistList p31]
[$HistListLen p32]
[IOindex p24]
[HiFiAccess p937]
[InteractiveFrame p23]

— defun undoInCore —

(defun |undoInCore| (n)
  (let (li vec p p1 val)
    (declare (special |$HistList| |$HistListLen| |IOindex| |HiFiAccess|)
$\text{25.23. \textit{Initialized History Variables}}$

```lisp
(defun undoChanges (li)
  (let (x)
    (declare (special $\text{HistList}$$\text{InteractiveFrame}$))
    (when (null (boot-equal (cdr li) $\text{HistList}$)) (undoChanges (cdr li)))
    (dolist (p1 (car li))
      (setq x (car p1))
      (dolist (p2 (cdr p1))
        (putHist x (car p2) (cdr p2) $\text{InteractiveFrame}$))))
)
```

---

**defun undoChanges**

[boot-equal p??]
undoChanges p835
[putHist p832]
$\text{HistList}$ p31
$\text{InteractiveFrame}$ p23

---

---

---
defun undoFromFile

[seq p??]
[exit p??]
[recordOldValue p833]
[recordNewValue p833]
[readHiFi p843]
[disableHist p845]
[putHist p832]
[assq p1200]
[updateHist p831]
[$InteractiveFrame p23]
[$HiFiAccess p937]

— defun undoFromFile —

(defun |undoFromFile| (n)
  (let (varl prop vec x p p1 val)
    (declare (special |$InteractiveFrame| |$HiFiAccess|))
    (do ((tmp0 (caar |$InteractiveFrame|) (cdr tmp0)) (tmp1 nil))
        ((or (atom tmp0)
            (progn (setq tmp1 (car tmp0)) nil)
            (progn
                (setq x (car tmp1))
                (setq varl (cdr tmp1))
                tmp1)
            nil))
        nil)
    (seq
      (exit
       (do ((tmp2 varl (cdr tmp2)) (p nil))
           ((or (atom tmp2) (progn (setq p (car tmp2)) nil)) nil)
        (seq
         (exit
          (progn
            (setq prop (car p))
            (setq val (cdr p))
            (when val
              (progn
                (when (null (eq x '%))
                  (|recordOldValue| x prop val))
                (when |HiFiAccess|
                  (|recordNewValue| x prop val))
                (rplacd p nil))))))))
    (do ((i 1 (1+ i)))
        ((> i n) nil)
    (setq vec
      (unwind-protect (cdr (|readHiFi| i)) (|disableHist|))))
(defun saveHistory (fn)
  (let ((|$seen| savefile inputfile saveStr n rec val)
      (declare (special |$seen| |$HiFiAccess| |$useInternalHistoryTable|
              |$internalHistoryTable|))
    (setq |$seen| (make-hash-table :test #'eq))
    (cond
      ((null |$HiFiAccess|)
The history facility is not on, so no information can be saved.

((and (null (null (null (null (null (null (null (null (null (null (null (null (null (null  

You must specify a file name to the history save command

You must specify a file name to the history save command

Can’t save the value of step number %1. You can re-generate this value by running the input file %2.

The saved history file is %1 .
defun restoreHistory

[setq p??]
[setq p??]
[identp p1197]
[throwKeyedMsg p??]
[makeHistFileName p819]
[putHist p832]
[makeInputFilename p1129]
[sayKeyedMsg p27]
[namestring p1190]
[clearSpad2Cmd p767]
.histFileName p820]
[histFileErase p860]
[$fcopy p??]
[rkeyids p??]
[readHiFi p843]
[disableHist p845]
[updateInCoreHist p832]
[setq p??]
[rempropI p??]
[clearCmdSortedCaches p769]
[$options p??]
[$internalHistoryTable p32]
[$HiFiAccess p937]
[$e p247]
[$useInternalHistoryTable p819]
[$InteractiveFrame p23]
[$oldHistoryFileName p818]

— defun restoreHistory —

(let ((|$options| fnq restfile curfile 1 oldInternal vec line x a))
(declare (special |$options| |$internalHistoryTable| |$HiFiAccess| |$e|
 |$useInternalHistoryTable| |$InteractiveFrame| |$oldHistoryFileName|)))
(cond
  ((null fn) (setq fnq |$oldHistoryFileName|)))
  (and (consp fn)
     (eq (qcdr fn) nil)
     (progn
      (setq fnq (qcar fn))
      t)
     (identp fnq))
     (setq fnq fnq))
  (t (|throwKeyedMsg| "%1 is not a valid filename for the history file." (cons fnq nil))))) ; invalid filename
(setq restfile (makeHistFileName fnq))
(if (null (makeInputFilename restfile))
 (sayKeyedMsg
  (format nil
    "History information cannot be restored from %1 because the file does not exist."
    (namestring restfile)) nil)
(progn
  (setq $options nil)
  (clearSpad2Cmd '(all))
  (setq curfile (histFileName))
  (histFileErase curfile)
  ($fcopy restfile curfile)
  (setq l (length (rkeyids curfile)))
  (setq $HiFiAccess t)
  (setq oldInternal $useInternalHistoryTable)
  (setq $useInternalHistoryTable nil)
  (when oldInternal (setq $internalHistoryTable nil))
  (do ((i 1 (1+ i)))
    (> i l) nil)
  (setq vec (unwind-protect (readHiFi i) (disableHist)))
  (when oldInternal
    (setq $internalHistoryTable
      (cons (cons i vec) $internalHistoryTable)))
  (setq line (car vec))
  (dolist (p1 (cdr vec))
    (setq x (car p1))
    (do ((tmp1 (cdr p1) (cdr tmp1)) (p2 nil))
      ((or (atom tmp1) (progn (setq p2 (car tmp1)) nil)) nil)
      (setq $InteractiveFrame
        (putHist x
          (car p2) (cdr p2) $InteractiveFrame)))
    (updateInCoreHist))
  (setq $e $InteractiveFrame)
  (do ((tmp2 (caar $InteractiveFrame) (cdr tmp2)) (tmp3 nil))
    ((or (atom tmp2)
      (progn
        (setq tmp3 (car tmp2))
        nil)) nil)
    (rempropI a '$localModemap)
    (rempropI a '$localVars)
    (rempropI a '$mapBody))
  (setq $IOindex (1+ l)))
(setq $useInternalHistoryTable| oldInternal)
(|sayKeyedMsg|
   "The workspace has been successfully restored from the history file %1 ."
   (cons (|namestring| restfile) nil))
(|clearCmdSortedCaches| nil))))

---

(defun setIOindex

[|IOindex p24|]

— defun setIOindex —

(defun |setIOindex| (n)
   (declare (special |$IOindex|))
   (setq |$IOindex| n))

---

(defun showInput

[|tab p??|]
[|readHiFi p843|]
[|disableHist p845|]
[|sayMSG p29|]

— defun showInput —

(defun |showInput| (mini maxi)
   (let (vec l)
      (do ((|ind| mini (+ |ind| 1)))
           ((> |ind| maxi) nil)
         (setq vec (unwind-protect (|readHiFi| |ind|) (|disableHist|)))
         (cond
            ((> 10 |ind|) (tab 2))
            ((> 100 |ind|) (tab 1))
            (t nil))
      (setq l (car vec))
      (if (stringp l)
         (|sayMSG| (list " [" |ind| "] " (car vec)))
         (progn
            (|sayMSG| (list " [" |ind| "] "))
            (do ((tmp0 l (cdr tmp0)) (ln nil))
               ((null l) ln))))
(or (atom tmp0) (progn (setq ln (car tmp0)) nil) (sayMSG (list " " ln))))

---

defun showInOut

[assq p1200]
[spadPrint p??]
[objValUnwrap p448]
[objMode p448]
[readHiFi p843]
[disableHist p845]
[sayMSG p29]

--- defun showInOut ---

(defun showInOut (mini maxi)
  (let (vec Alist triple)
    (do ((ind mini (+ ind 1)))
        ((> ind maxi) nil)
      (setq vec (unwind-protect (readHiFi ind) (disableHist)))
      (sayMSG (cons (car vec) nil))
      (cond
        ((setq Alist (assq '%' (cdr vec)))
         (setq triple (cdr (assq 'value (cdr Alist))))
         (setq $IOindex ind)
         (spadPrint (objValUnwrap triple) (objMode triple)))))))

---

defun fetchOutput

[boot-equal p??]
[getI p??]
[throwKeyedMsg p??]
[readHiFi p843]
[disableHist p845]
[assq p1200]

--- defun fetchOutput ---

(defun fetchOutput (n)
  (let (vec Alist val)
25.23. **INITIALIZED HISTORY VARIABLES**

```lisp
(defun readHiFi |\$HiFiAccess|)
  (setq n
    (cond
      ((minusp n) (+ |\$IOindex| n))
      (t n))
    (cond
      (>= n |\$IOindex|)
        (throwKeyedMsg
          "You have not reached step %1, and so its value cannot be supplied."
          (cons n nil))
      (> 1 n)
        (throwKeyedMsg
          "Cannot supply value for step %1b because 1 is the first step."
          (cons n nil)) ; only nonzero steps
      (t
        (setq vec (unwind-protect (readHiFi n) (disableHist)))
        (cond
          ((setq Alist (assq '%' (cdr vec)))
            (cond
              ((setq val (cdr (assq '|value| (cdr Alist))))
                val)
              (t
                (throwKeyedMsg "Step %1 has no value." (cons n nil))))))
      (t (throwKeyedMsg "The history facility is not on, so you cannot use %%." nil))) ; history not on
```

---

**Read the history file using index n**

[assoc p??]
[keyedSystemError p??]
[qcdr p??]
[rdefiostream p??]
[histFileName p820]
[spadrread p847]
[object2Identifier p??]
[rshut p??]
[\$useInternalHistoryTable p819]
[\$internalHistoryTable p32]
(defun |readHiFi| (n)
"Read the history file using index n"
(let (pair HiFi vec)
   (declare (special |$useInternalHistoryTable| |$internalHistoryTable|))
   (if |$useInternalHistoryTable|
      (progn
       (setq pair (|assoc| n |$internalHistoryTable|))
       (if (atom pair)
           (|keyedSystemError| "Missing element in internal history table." nil)
           (setq vec (qcdr pair))))
      (progn
       (setq HiFi
            (rdefiostream
             (cons
              '(mode . input)
              (cons
               (cons 'file (|histFileName|)) nil))))
       (setq vec (spadrread (|object2Identifier| n) HiFi))
       (rshut HiFi)))
   vec))

— defun writeHiFi —

Write information of the current step to history file

[rdefiostream p??]  
[histFileName p820]  
[spadrwrite p847]  
[object2Identifier p??]  
[rshut p??]  
|$useInternalHistoryTable p819|  
|$internalHistoryTable p32|  
|$IOindex p24|  
|$HistRecord p32|  
|$currentLine p??|

(defun |writeHiFi| ()
"Writes information of the current step to history file"
(let (HiFi)
   (declare (special |$useInternalHistoryTable| |$internalHistoryTable||
            |$IOindex| |$HistRecord| |$currentLine|)))
   (if |$useInternalHistoryTable|
      (setq |$internalHistoryTable|
            (cons
             (cons |$IOindex|)
             (cons |$HistRecord|))))
   vec))
25.23. **INITIALIZED HISTORY VARIABLES**

```
(cons |$currentLine| |$HistRecord|)
($internalHistoryTable|)
(progn
  (setq HiFi
    (rdefiostream
      (cons
        '(mode . output)
        (cons (cons 'file (|histFileName|)) nil))))
  (spadrwrite (|object2Identifier| |$IOindex|
    (cons |$currentLine| |$HistRecord|) HiFi)
  (rshut HiFi))))
```

---

**Disable history if an error occurred**

```
(histFileErase p860)
(histFileName p820)
($HiFiAccess p937)
```

---

---

---

**defun disableHist**

```
(defun disableHist ()
  "Disable history if an error occurred"
  (declare (special |$HiFiAccess|))
  (cond
    ((null |$HiFiAccess|)
      (histFileErase (histFileName))
      (t nil)))
```

---

**defun writeHistModesAndValues**

```
(get p??)
(putHist p832)
($InteractiveFrame p23)
```

---

---

---

---

---

---

---

```
(defun writeHistModesAndValues ()
  (let (a x)
    (declare (special |$InteractiveFrame|))
    (do ((tmp0 (caar |$InteractiveFrame|) (cdr tmp0)) (tmlp nil))
      ((or (atom tmp0)
```
Lisplib output transformations

Some types of objects cannot be saved by LISP/VM in lisplibs. These functions transform an object to a writable form and back.

**defun spadrwrite0**

|safeWritify p849|
|rwrite p846|

--- defun spadrwrite0 ---

(defun spadrwrite0 (vec item stream)
  (let (val)
    (setq val (|safeWritify| item))
    (if (eq val '|writifyFailed|)
      val
      (progn
        (|rwrite| vec val stream)
        item)))))

---

**defun Random write to a stream**

|rwrite p846|
|pname p1195|
|identp p1197|
— defun rwrite —

(defun rwrite (key val stream)
  (when (identp key) (setq key (pname key)))
  (rwrite key val stream))

— defun spadrwrite —

(defun spadrwrite (vec item stream)
  (let (val)
    (setq val (spadrwrite0 vec item stream))
    (if (eq val '|writifyFailed|)
      (throwKeyedMsg "The value specified cannot be saved to a file." nil)
      item)))

— defun spadrread —

(defun spadrread (vec stream)
  (dewritify (rread vec stream nil)))

— defun Random read a key from a stream

RREAD takes erroval to return if key is missing
---

**defun rread**

```
(defun rread (key rstream errorval)
  (when (identp key) (setq key (pname key)))
  (rread key rstream errorval))
```  

---

**defun unwritable?**

```
(defun unwritable? (ob)
  (cond
   ((or (consp ob) (vecp ob)) nil)
   ((or (compiled-function-p ob) (hash-table-p ob)) t)
   ((or (placep ob) (readtablep ob)) t)
   ((floatp ob) t)
   (t nil)))
```  

---

**defun writifyComplain**

Create a full isomorphic object to be saved in a lisplib. Note that `dewritify(writify(x))` preserves UEQUALity of hashtables. HASHTABLEs go both ways. READTABLEs cannot presently be transformed back.  

```
(defun writifyComplain (s)
  (declare (special #$writifyComplained))
  (unless #$writifyComplained
    (setq #$writifyComplained t)
    (sayKeyedMsg
     (format nil
       "A value containing a %1 is being saved in a history file or a ~
       compiled input file INLIB. This type is not yet usable in other ~")
     s))
```  

---
history operations. You might want to issue \texttt{)history }\texttt{off")} 
(list s)))) ; cannot save value

---

defun safeWritify

[ writifyTag p?? ]
[ writify p852 ]

---

(defun safeWritify)

(defun safeWritify (ob)
  (catch 'writifyTag (writify ob)))

---

defun writify,writifyInner

[ writifyTag p?? ]
[ seq p?? ]
[ exit p?? ]
[ hget p1194 ]
[ qcar p?? ]
[ qcdr p?? ]
[ spadClosure? p853 ]
[ writify,writifyInner p849 ]
[ hput p1194 ]
[ qrplaca p?? ]
[ qrplacd p?? ]
[ vecp p?? ]
[ isDomainOrPackage p1061 ]
[ mkEvalable p1075 ]
[ devaluate p?? ]
[ qvmaxindex p?? ]
[ qsetvelt p?? ]
[ qvelt p?? ]
[ constructor? p?? ]
[ hkeys p1195 ]
[ hashtable-class p?? ]
[ placep p1227 ]
[ boot-equal p?? ]
[ $seen p?? ]
— defun writify,writifyInner —

(defun writify,writifyInner (ob)
  (prog (e name tmp1 tmp2 tmp3 x qcar qcdr d n keys nob)
    (declare (special $seen $NonNullStream $NullStream))
    (return
      (seq
        (when (null ob) (exit nil))
        (when (setq e (hget $seen ob)) (exit e))
        (when (consp ob)
          (exit
            (seq
              (setq qcar (qcar ob))
              (setq qcdr (qcdr ob))
              (when (setq name (spadClosure? ob))
                (exit
                  (seq
                    (setq d (writify,writifyInner (qcdr ob)))
                    (setq nob
                      (cons 'writified!!
                        (cons 'spadclosure
                          (cons d (cons name nil))))))
                    (hput $seen ob nob)
                    (hput $seen nob nob)
                    (exit nob))))))
        (when
          (and
            (and (consp ob)
              (eq (qcar ob) 'lambda-closure)
              (progn
                (setq tmp1 (qcdr ob))
                (and (consp tmp1)
                  (progn
                    (setq tmp2 (qcar tmp1))
                    (and
                      (consp tmp2)
                      (progn
                        (setq tmp3 (qcdr tmp2))
                        (and (consp tmp3)
                          (progn
                            (setq x (qcar tmp3))
                            t))))))))
          (throw '|writifyTag| '|writifyFailed|))
          (setq nob (cons qcar qcdr))
          (hput $seen ob nob)
          (hput $seen nob nob))}
25.23. **INITIALIZED HISTORY VARIABLES**

```lisp
(setq qcar (writifyInner qcar))
(setq qcdr (writifyInner qcdr))
(qrplaca nob qcar)
(qrplacd nob qcdr)
(exit nob)))

(when (vecp ob)
  (exit
   (setq d (devaluate ob)))
  (setq nob (list 'writified!! 'devaluated (writifyInner d)))
  (hput $seen ob nob)
  (hput $seen nob nob)
  (exit nob))

(when (isDomainOrPackage ob)
  (setq n (qvmindex ob))
  (setq nob (make-array (1+ n)))
  (hput $seen ob nob)
  (hput $seen nob nob)
  (do ((i 0 (=! i)))
       ((> i n) nil)
    (qsetvelt nob i (writifyInner (qvelt ob i))))
  (exit nob)))

(when (eq ob 'writified!!)
  (exit
   (cons 'writified!! (cons 'self nil))))

(when (constructor? ob)
  (exit ob))

(when (compiled-function-p ob)
  (exit
   (throw '|writifyTag| '|writifyFailed|)))

(when (hash-table-p ob)
  (setq nob (cons 'writified!! nil))
  (hput $seen ob nob)
  (hput $seen nob nob)
  (setq keys (hkeys ob))
  (qrplacd nob
    (cons
      'hashtable
    (cons
      (hashtable-class ob)
    (cons
      (writifyInner keys)
    (cons
      (prog (tmp0)
        (setq tmp0 nil)
        (return
          (do ((tmp1 keys (cdr tmp1)) (k nil))
              ((or (atom tmp1)
                (progn
                  (setq k (car tmp1)))))
            nil)))))
```
CHAPTER 25. SYSTEM COMMAND HANDLING

nil))
    (nreverse0 tmp0))
    (setq tmp0
        (cons ([writify,writifyInner] (hget ob k)) tmp0))))
    nil))))))
(when (placep ob)
    (setq nob (cons 'writified!! (cons 'place nil)))
    (hput $seen ob nob)
    (hput $seen nob nob)
    (exit nob))
(when (readtablep ob)
    (exit
      (throw '|writifyTag| '|writifyFailed|)))
(when (stringp ob)
    (exit
      (seq
        (when (eq ob $NullStream)
          (exit
            (cons 'writified!! (cons 'nullstream nil))))
        (when (eq ob $NonNullStream)
          (exit
            (cons 'writified!! (cons 'nonnullstream nil))))
        (exit ob))))
(when (floatp ob)
    (exit
      (seq
        (when (boot-equal ob (read-from-string (princ-to-string ob)))
          (exit ob))
        (exit
          (cons 'writified!!
            (cons 'float
              (cons ob
                (multiple-value-list (integer-decode-float ob))))))))
    (exit ob))))))

defun writify

[ScanOrPairVec p858]
[function p??]
[writify,writifyInner p849]
[$seen p??]
[$writifyComplained p??]

—— defun writify ——
(defun |writify| (ob)
  (let ([|$seen|] [|$writifyComplained|])
    (declare (special [|$seen|] [|$writifyComplained|]))
    (if (null (|ScanOrPairVec| #'|unwritable?| ob))
      ob
      (progn
        (setq [|$seen|] (make-hash-table :test #'eq))
        (setq [|$writifyComplained|] nil)
        ([|writify,writifyInner|] ob))))

---

defun spadClosure?
[qcar p??]
[bpiname p??]
[qcdr p??]
[vecp p??]

— defun spadClosure? —

(defun |spadClosure?| (ob)
  (let (fun name vec)
    (setq fun (qcar ob))
    (if (null (setq name (bpiname fun)))
      nil
      (progn
        (setq vec (qcdr ob))
        (if (null (vecp vec))
          nil
          name))))

---

defvar $NonNullStream

— initvars —

(defun |$NonNullStream| ["NonNullStream"]

---
defvar $NullStream

— initvars —

(defvar |$NullStream| "NullStream")

defun dewritify,dewritifyInner

[seq p??]
[exit p??]
[hget p1194]
[intp p??]
[gensymmer p??]
[error p??]
[poundsign p??]
[hput p1194]
[dewritify,dewritifyInner p854]
[concat p1197]
[vmread p1228]
[make-instream p1127]
[qcar p??]
[qcdr p??]
[qrplaca p??]
[qrplacd p??]
[vecp p??]
[qvmaxindex p??]
[qsetqvelt p??]
[qvelt p??]
[$seen p??]
[$NullStream p854]
[$NonNullStream p854]

— defun dewritify,dewritifyInner —

(defun |dewritify,dewritifyInner| (ob)
    (prog (e type oname f vec name tmp1 signif expon sign fval qcar qcdr n nob)
        (declare (special |$seen| |$NullStream| |$NonNullStream|))
        (return
            (seq
                (when (null ob)
                    (exit nil))
                (when (setq e (hget |$seen| ob))))
(exit e))
(when (and (consp ob) (eq (car ob) 'writified!!))
  (exit
    (setq type (elt ob 1))
    (when (eq type 'self)
      (exit 'writified!!))
    (when (eq type 'bpi)
      (exit
        (setq oname (elt ob 2))
        (setq f
          (seq
            (when (integerp oname) (exit (eval (gensymmer oname)))))
            (exit (symbol-function oname))))
      (when (null (compiled-function-p f))
        (error "A required BPI does not exist."))
      (when (and (> (length ob) 3) (not (equal (sxhash f) (elt ob 3))))
        (error "A required BPI has been redefined."))
      (hput |$seen| ob f)
      (exit f))))
  (when (eq type 'hashtable)
    (exit
      (setq nob (make-hash-table :test #'equal))
      (hput |$seen| ob nob)
      (hput |$seen| nob nob)
      (do ((tmp0 (elt ob 3) (cdr tmp0))
           (k nil)
           (tmp1 (elt ob 4) (cdr tmp1))
           (e nil))
          ((or (atom tmp0)
                (progn
                  (setq k (car tmp0))
                  nil)
                (atom tmp1)
                (progn
                  (setq e (car tmp1))
                  nil))
           nil)
      (seq
        (exit
          (hput nob (|dewritify,dewritifyInner| k)
            (|dewritify,dewritifyInner| e))))
      (exit nob))))
  (when (eq type 'devaluated)
    (exit
      (setq nob (eval (|dewritify,dewritifyInner| (elt ob 2))))
      (hput |$seen| ob nob))
(hput |$seen| nob nob)
(exit nob)))

(when (eq type 'spadclosure)
(exit
(seq
(setq vec (|dewritify,dewritifyInner| (elt ob 2)))
(setq name (ELT ob 3))
(when (null (fboundp name))
(exit
(|error|
 (concat "undefined function: " (symbol-name name))))))
(setq nob (cons (symbol-function name) vec))
(hput |$seen| ob nob)
(hput |$seen| nob nob)
(exit nob))))

(when (eq type 'place)
(exit
(seq
(setq nob (vmread (make-instream nil)))
(hput |$seen| ob nob)
(hput |$seen| nob nob)
(exit nob))))

(when (eq type 'readtable)
(exit (|error| "Cannot de-writify a read table.")))

(when (eq type 'nullstream)
(exit |$NullStream|))

(when (eq type 'nonnullstream)
(exit |$NonNullStream|))

(when (eq type 'float)
(exit
(seq
(progn
 (setq tmp1 (cddr ob))
 (setq fval (car tmp1))
 (setq signif (cadr tmp1))
 (setq expon (caddr tmp1))
 (setq sign (cadddr tmp1))
 tmp1)
 (setq fval (scale-float (float signif fval) expon))
 (when (minusp sign)
 (exit (- fval)))
 (exit fval))))

(exit (|error| "Unknown type to de-writify.")))))

(when (consp ob)
(exit
(seq
(setq qcar (qcar ob))
(setq qcdr (qcdr ob))
(setq nob (cons qcar qcdr))
(hput |$seen| ob nob)
25.23. INITIALIZED HISTORY VARIABLES

```lisp
(hput |$seen| nob nob)
(qrplaca nob (|dewritify,dewritifyInner| qcar))
(qrplacd nob (|dewritify,dewritifyInner| qcdr))
(exit nob))))
(when (vecp ob)
  (exit
    (seq
      (setq n (qvmaxindex ob))
      (setq nob (make-array (1+ n)))
      (hput |$seen| ob nob)
      (hput |$seen| nob nob)
      (do ((i 0 (1+ i)))
        ((> i n) nil)
        (seq
          (exit
            (setqvelt nob i
              (|dewritify,dewritifyInner| (qvelt ob i))))))
      (exit nob))))
(exit ob))))
```

---

defun dewritify

[ScanOrPairVec p858]
[function p??]
[dewritify,dewritifyInner p854]
|$seen p??|

— defun dewritify —

```lisp
(defun |dewritify| (ob)
  (let ((|$seen|))
    (declare (special |$seen|))
    (if (null (|ScanOrPairVec| #'(lambda (a) (eq a 'writified!!)) ob))
      ob
      (progn
        (setq |$seen| (make-hash-table :test '#eq))
        (|dewritify,dewritifyInner| ob))))
```

---

defun ScanOrPairVec,ScanOrInner

[ScanOrPairVecAnswer p??]
[hget p1194]
defun ScanOrPairVec,ScanOrInner

(defun |ScanOrPairVec,ScanOrInner| (f ob)
  (declare (special |$seen|))
  (when (hget |$seen| ob) nil)
  (when (consp ob)
    (hput |$seen| ob t)
    (|ScanOrPairVec,ScanOrInner| f (qcar ob))
    (|ScanOrPairVec,ScanOrInner| f (qcdr ob)))
  (when (vecp ob)
    (hput |$seen| ob t)
    (do ((tmp0 (- (#| ob) 1)) (i 0 (1+ i)))
      (> i tmp0) nil)
      (|ScanOrPairVec,ScanOrInner| f (elt ob i)))
  (when (funcall f ob) (throw '|ScanOrPairVecAnswer| t))
  nil)

defun ScanOrPairVec

(defun |ScanOrPairVec| (f ob)
  (let (|$seen|)
    (declare (special |$seen|))
    (setq |$seen| (make-hash-table :test #'eq))
    (catch '|ScanOrPairVecAnswer| (|ScanOrPairVec,ScanOrInner| f ob))))
defun gensymInt

| defun gensymInt |
| (defun gensymInt (g)
  (let (p n)
    (if (null (gensymp g))
      (|error| "Need a GENSYM")
      (progn
        (setq p (pname g))
        (setq n 0)
        (do ((tmp0 (- (|#| p) 1)) (i 2 (1+ i)))
          ((> i tmp0) nil)
          (setq n (+ (* 10 n) (|charDigitVal| (elt p i))))))
    n))))

defun charDigitVal

| defun charDigitVal |
| (defun charDigitVal (c)
  (let (digits n)
    (setq digits "0123456789")
    (setq n (- 1))
    (do ((tmp0 (- (|#| digits) 1)) (i 0 (1+ i)))
      ((or (> i tmp0) (null (minusp n))) nil)
      (if (char= c (elt digits i))
        (setq n i)
        nil))
    (if (minusp n)
      (|error| "Character is not a digit")
    n))

|— defun gensymInt —|
|— defun charDigitVal —|
defun histFileErase

--- defun histFileErase ---

(defun |histFileErase| (file)
  (when (probe-file file) (delete-file file)))

---
25.24 \textit{\texttt{\texttt{\textbackslash include}}} Command

\texttt{\texttt{\textbackslash include}}} man page

--- \texttt{\texttt{\texttt{\texttt{\textbackslash include}}}\texttt{.help}} ---

User Level Required: \texttt{interpreter}

Command Syntax:

\texttt{\texttt{\texttt{\texttt{\textbackslash include}}}} filename

Command Description:

The \texttt{\texttt{\texttt{\texttt{\textbackslash include}}}} command can be used in \texttt{\texttt{\texttt{\texttt{.input}}} files to place the contents of another file inline with the current file. The path can be an absolute or relative pathname.

\begin{verbatim}
defun ncloopInclude1 (ncloopIncFileName p
[ncloopInclude p]
[ncloopIncFileName p]

defun ncloopInclude1 (name n)
(let (a)
  (if (setq a (ncloopIncFileName name))
      (ncloopInclude a n)
    n)))

---

Returns the first non-blank substring of the given string

\begin{verbatim}
defun ncloopIncFile (

\begin{verbatim}
defun ncloopIncFile (string)
"Returns the first non-blank substring of the given string"
(let (fn)
  (unless (setq fn (|incFileName| string))
    (write-line (concat string " not found")))
  fn))

Open the include file and read it in

The ncloopInclude0 function is part of the parser and lives in int-top.boot.
[ncloopInclude0 p297]

— defun ncloopInclude —

(defun |ncloopInclude| (name n)
  "Open the include file and read it in"
  (with-open-file (st name) (|ncloopInclude0| st name n)))

Return the include filename

Given a string we return the first token from the string which is the first non-blank substring.
[incBiteOff p862]

— defun incFileName —

(defun |incFileName| (x)
  "Return the include filename"
  (car (|incBiteOff| x)))

Return the next token

Takes a sequence and returns the a list of the first token and the remaining string characters.
If there are no remaining string characters the second string is of length 0. Effectively it
"bites off" the first token in the string. If the string only 0 or more blanks it returns nil.
— defun incBiteOff —

(defun |incBiteOff| (x)
"Return the next token"
(let (blank nonblank)
  (setq x (string x))
  (when (setq nonblank (position #\space x :test-not #'char=))
    (setq blank (position #\space x :start nonblank))
    (if blank
      (list (subseq x nonblank blank) (subseq x blank))
      (list (subseq x nonblank) ""))))
25.25  )library Command

    library man page

    — library.help —

====================================================================
A.14.  )library
====================================================================

User Level Required:  interpreter

Command Syntax:

- )library libName1  [libName2 ...]
- )library )dir dirName
- )library )only objName1  [objlib2 ...]
- )library )noexpose

Command Description:

This command replaces the )load system command that was available in AXIOM
releases before version 2.0. The )library command makes available to AXIOM
the compiled objects in the libraries listed.

For example, if you )compile dopler.spad in your home directory, issue )library
dopler to have AXIOM look at the library, determine the category and domain
constructors present, update the internal database with various properties of
the constructors, and arrange for the constructors to be automatically loaded
when needed. If the )noexpose option has not been given, the constructors
will be exposed (that is, available) in the current frame.

If you compiled a file you will have an NRLIB present, for example,
DOPLER.NRLIB, where DOPLER is a constructor abbreviation. The command
)library DOPLER will then do the analysis and database updates as above.

To tell the system about all libraries in a directory, use )library )dir
dirName where dirName is an explicit directory. You may specify ‘‘.’’ as the
directory, which means the current directory from which you started the
system or the one you set via the )cd command. The directory name is required.

You may only want to tell the system about particular constructors within a
library. In this case, use the )only option. The command )library dopler
)only Test1 will only cause the Test1 constructor to be analyzed, autoloaded,
etc..

Finally, each constructor in a library are usually automatically exposed when
the )library command is used. Use the )noexpose option if you not want them
exposed. At a later time you can use \texttt{set expose add constructor} to expose any hidden constructors.

Note for AXIOM beta testers: At various times this command was called \texttt{local} and \texttt{with} before the name \texttt{library} became the official name.

Also See:
\begin{itemize}
\item \texttt{cd}
\item \texttt{compile}
\item \texttt{frame}
\item \texttt{set}
\end{itemize}

\footnote{\texttt{cd} (25.11 p 764) \texttt{frame} (2.5 p 10) \texttt{set} (25.51 p 1013)}
25.26  )license Command

    license man page

    — license.help —

==============================================================
A.15. )license
==============================================================

Command Syntax:
    - )license

Command Description:

This command displays the Axiom license.

Also See:
    o )trademark

———

(defun license

[obey p??]
[concat p1197]
[getenviron p255]

—— defun license ——

(defun |license| (l)
  (declare (ignore l))
  (obey (concat "cat " (getenv "AXIOM") "/doc/spadhelp/copyright.help")))

———
25.27 )lisp Command

lisp man page

— lisp.help —

====================================================================
A.15. )lisp
====================================================================

User Level Required: development

Command Syntax:

- )lisp [lispExpression]

Command Description:

This command is used by AXIOM system developers to have single expressions
evaluated by the Lisp system on which AXIOM is built. The lispExpression is
read by the Lisp reader and evaluated. If this expression is not complete
(unbalanced parentheses, say), the reader will wait until a complete
expression is entered.

Since this command is only useful for evaluating single expressions, the )fin
command may be used to drop out of AXIOM into Lisp.

Also See:

o )system
o )boot
o )fin

———

This command is in the list of $noParseCommands 25.1 which means that its arguments are
passed verbatim. This will eventually result in a call to the function handleNoParseCommands 25.2

11 “system” (25.56 p 1041) “boot” (25.4 p 758) “fin” (25.20 p 808)
25.28  )ltrace Command

ltrace man page

--- ltrace.help ---

==============================================
A.17. )ltrace
==============================================

User Level Required: development

Command Syntax:

This command has the same arguments as options as the )trace command.

Command Description:

This command is used by AXIOM system developers to trace Lisp or BOOT functions. It is not supported for general use.

Also See:
  o )boot
  o )lisp
  o )trace

---

12
defun The top level )ltrace function

[trace p1045]

--- defun ltrace ---

(defun ltrace (arg) (ltrace arg))

---

12 “boot” (25.4 p 758) “lisp” (25.27 p 867) “trace” (25.59 p 1045)
25.29  )pquit Command

pquit man page

--- pquit.help ---

====================================================================
A.18. )pquit
====================================================================

User Level Required: interpreter

Command Syntax:
- )pquit

Command Description:

This command is used to terminate AXIOM and return to the operating system. Other than by redoing all your computations or by using the )history )restore command to try to restore your working environment, you cannot return to AXIOM in the same state.

)pquit differs from the )quit in that it always asks for confirmation that you want to terminate AXIOM (the ‘p’ is for ‘protected’). When you enter the )pquit command, AXIOM responds

Please enter y or yes if you really want to leave the interactive environment and return to the operating system:

If you respond with y or yes, you will see the message

You are now leaving the AXIOM interactive environment.
Issue the command axiom to the operating system to start a new session.

and AXIOM will terminate and return you to the operating system (or the environment from which you invoked the system). If you responded with something other than y or yes, then the message

You have chosen to remain in the AXIOM interactive environment.

will be displayed and, indeed, AXIOM would still be running.

Also See:
o )fin
o )history
o )close
o )quit
The top level pquit command

(defun pquit ()
  "The top level pquit command"
  (pquitSpad2Cmd))

The top level pquit command handler

(defun pquitSpad2Cmd ()
  "The top level pquit command handler"
  (let ((quitCommandType 'protected))
    (declare (special quitCommandType))
    (quitSpad2Cmd))))

This command is in the list of $noParseCommands$ which means that its arguments are passed verbatim. This will eventually result in a call to the function handleNoParseCommands.
25.30  )QUIT COMMAND

25.30  )quit Command

quit man page

— quit.help —

====================================================================
A.19. )quit
====================================================================

User Level Required: interpreter

Command Syntax:

- )quit
- )set quit protected | unprotected

Command Description:

This command is used to terminate AXIOM and return to the operating system. Other than by redoing all your computations or by using the )history )restore command to try to restore your working environment, you cannot return to AXIOM in the same state.

)quit differs from the )pquit in that it asks for confirmation only if the command

)set quit protected

has been issued. Otherwise, )quit will make AXIOM terminate and return you to the operating system (or the environment from which you invoked the system).

The default setting is )set quit protected so that )quit and )pquit behave in the same way. If you do issue

)set quit unprotected

we suggest that you do not (somehow) assign )quit to be executed when you press, say, a function key.

Also See:
  o )fin
  o )history
  o )close
  o )pquit
  o )system
The top level quit command

(defun quit ()
"The top level quit command"
(quitSpad2Cmd))

The top level quit command handler

(defun quitSpad2Cmd ()
"The top level quit command handler"
(declare (special $quitCommandType))
(if (eq $quitCommandType 'protected)
(let (x)
(setq x
(upcase
(queryUserKeyedMsg
(format nil
"Please enter y or yes if you really want to leave the interactive ~
environment and return to the operating system."
nil)))
(member (string2id-n x 1) '(y yes)) (leaveScratchpad))
(sayKeyedMsg
"You have chosen to remain in the Axiom interactive environment." nil)
(tersyscommand)))
25.30.  \textit{QUIT COMMAND}

\begin{verbatim}
Leave the Axiom interpreter

\texttt{(defun \texttt{leaveScratchpad} ()
   "Leave the Axiom interpreter"
   (bye))}
\end{verbatim}

This command is in the list of \texttt{$noParseCommands$} which means that its arguments are passed verbatim. This will eventually result in a call to the function \texttt{handleNoParseCommands}.
25.31  )read Command

    read man page

    — read.help —

====================================================================
A.20.  )read
====================================================================

User Level Required:  interpreter

Command Syntax:

    -  )read [fileName]
    -  )read [fileName] [)quiet] [)ifthere]

Command Description:

This command is used to read .input files into AXIOM. The command

    )read matrix.input

will read the contents of the file matrix.input into AXIOM. The ‘’.input’
file extension is optional. See the AXIOM User Guide index for more
information about .input files.

This command remembers the previous file you edited, read or compiled. If you
do not specify a file name, the previous file will be read.

The )ifthere option checks to see whether the .input file exists. If it does
not, the )read command does nothing. If you do not use this option and the
file does not exist, you are asked to give the name of an existing .input
file.

The )quiet option suppresses output while the file is being read.

Also See:
    o  )compile
    o  )edit
    o  )history

15

15 “edit” (25.19 p 804) “history” (25.23 p 821)
defun The )read command

(defun |readSpad2Cmd| p875)

    — defun read —

(defun |read| (arg) (|readSpad2Cmd| arg))

———

defun Implement the )read command

(defun |readSpad2Cmd|)

    — defun readSpad2Cmd —

(defun |readSpad2Cmd| (arg)
  (prog (|$InteractiveMode| fullopt ifthere quiet ef devFTs fileTypes
  11 ft upft fs)
    (declare (special |$InteractiveMode| $findfile |$UserLevel| |$options|
    /editfile))
    (setq |$InteractiveMode| t)
    (dolist (opt |$options|)
      (setq fullopt
        (|selectOptionLC| (car opt) '(|quiet| |test| |ifthere|) '|optionError|)
        (cond
          ((eq fullopt '|ifthere|) (setq ifthere t))
          ((eq fullopt '|quiet|) (setq quiet t)))
      (setq ef (pathname /editfile)))
    (setq fullopt
      (|selectOptionLC| (car opt) '(|quiet| |test| |ifthere|) '|optionError|)
      (cond
        ((eq fullopt '|ifthere|) (setq ifthere t))
        ((eq fullopt '|quiet|) (setq quiet t))))
    (setq ef (pathname /editfile)))
(when (eq (|pathnameTypeId| ef) 'spad)
  (setq ef (|makePathname| (|pathnameName| ef) "*" "*"))
(if arg
  (setq arg (|mergePathnames| (|pathname| arg) ef))
  (setq arg ef))
(setq devFTs ('"input" "INPUT" "boot" "BOOT" "lisp" "LISP")
(setq fileTypes
  (cond
    ((eq |$UserLevel| '|interpreter|) ('"input" "INPUT"))
    ((eq |$UserLevel| '|compiler|) ('"input" "INPUT"))
    (t devFTs))
  (setq ll ($findfile arg fileTypes))
  (unless ll
    (if ifthere
      (return nil)
      (|throwKeyedMsg| "The file %1 is needed but does not exist." list (|namestring| arg))))
  (setq ll (|pathname| ll))
  (setq ft (|pathnameType| ll))
  (setq upft (upcase ft))
  (cond
    ((null (|member| upft fileTypes))
      (setq fs (|namestring| arg))
      (|throwKeyedMsg| (format nil "You cannot read the file %1 because your user-level is not high enough. For more information about your user-level, issue )set userlevel.") list fs))
    (|throwKeyedMsg| (format nil "You cannot read the file %1 because it is not suitable for reading by Axiom. Note that files with file extension .spad can now only be compiled with the )compile system command.") list fs))))
(t
  (setq /editfile ll)
  (when (string= upft "BOOT") (setq |$InteractiveMode| nil))
  (/read ll quiet)))))

defun /read

[/read /rf (vol9)]
[/read /rq (vol9)]
[/editfile p781]
— defun /read —

(defun /read (l q)
  (declare (special /editfile))
  (setq /editfile l)
  (cond
    (q (/rq))
    (t (/rf))
  )
  (flag |boot-NewKEY| 'key)
  (|terminateSystemCommand|)
  (|spadPrompt|))
25.32  )regress Command

regress man page

— regress.help —

====================================================================
A.18.  )regress
====================================================================

User Level Required:  interpreter

Command Syntax:

- )regress fileName

Command Description:

The )regress command will run the regress function that was compiled as part of the lisp image build process. This function expects an input filename, possibly containing a path prefix.

If the filename contains a period then we consider it a fully formed filename, otherwise we append ‘.output’, which is the default file extension.

)regress matrix
)regress matrix.output
)regress /path/to/file/matrix
)regress /path/to/file/matrix.output

will test the contents of the file matrix.output.

The idea behind regression testing is to check that the results we currently get match the results we used to get. In order to do that we create input files with a special comment format that contains the prior results. These are easy to create as all you need to do is run the Axiom function, capture the results, and turn them input specially formed comments using the -- comment.

A regression file caches the result of an Axiom function so we can automate the testing process. It is a file of many tests, each with their own output.

The regression file format uses the Axiom -- comment syntax to keep a copy of the expected output from an Axiom command. This expected output is compared character by character against the actual output.
The regression file is broken into numbered blocks, delimited by a --S for the beginning and a --E for the end. The total number of blocks is also given so missing or failed tests also raise an error.

There are 4 special kinds of -- comments in regression files:

--S n of M  this is test n of M tests in this file
--E n        this marks the end of test n
--R any output  this marks the actual expected output line
--I any output  this line is compared but ignored

A regression test file looks like:

)set break resume
)spool foo.output
)set message type off
)clear all

--S 1 of 3
2+3
--R  this is the exact Axiom output
--R  (1) 5
--E 1

--S 2 of 3
2+3
--R  this should fail to match
--R  (2) 7
--E 2

--S 3 of 3
2+3
--R  this fails to match but we
--I  (3) 7  use --I to ignore this line
--E 3

We can now run this file with

)read foo.input

Note that when this file is run it will create a spool file called "foo.output" because of the lines:

)spool foo.output
)spool

The "foo.output" file contains the console image of the result. It will look like:

(1) | 5
--R
--R (1) 5
--E 1

--S 2 of 3
2+3

(2) | 5
--R
--R (2) 7
--E 2

--S 3 of 3
2+3

(3) | 5
--R
--I (3) 7
--E 3

)spool

This "foo.output" file can now be checked using the )regress command.

When we run the )regress foo.output we see:

testing foo
passed foo 1 of 3
MISMATCH
expected:" (2) 7"
got:" (2) 5"
FAILED foo 2 of 2
passed foo 3 of 3
regression result FAILED 1 of 3 stanzas file foo

Tests either pass or fail. A passing test generates the message:

passed foo 1 of 3

A failing test will give a reversed printout of the expected vs actual output as well as a FAILED message, as in:

MISMATCH
The regress function details

This is the regression test mechanism. The input files have been rewritten to have a standard structure. This fixed format identifies the tests within a file. Each test is run and any mismatch between the actual and expected results is reported.

In order to regression test axiom results we created a standard file format. This format has 3 kinds of markers:

- “.S” marker which must have a integer test number
- “.R” marker lines, one per expected output from axiom
- “.E” marker which has an integer matching the preceeding “.S”
- “.I” marker ignores the line, useful for gensyms and random

Because these markers use Axiom’s standard comment prefix they are valid lines in input files and are ignored by the “)read” command. They are simply copied to the output file.
This allows us to include the expected output in the output file so we can compare what
Axiom computes with what it should compute.

To create these regression files all you need to do is create an input file and run it through
Axiom. Then, for each test case in the file you mark it up by putting a “—S number” before
the Axiom input line. You put “—R” prefixes on each line of Axiom output, including the
blank lines. Then you put a “—E number” line after the last output line, usually the Type:
line. This newly marked-up input file is now a regression test.

To actually run the regression test you simply include the marked up the input file in the
src/input subdirectory. This file will automatically be run at build time and any failing
tests will be marked. This code will ignore any input that does not contain proper regression
markups.

Ideally the regression test files should be pamphlet files that explain the content and purpose
of each regression test case.

Thus you run the marked-up input file foo.input and spool the result to foo.output and
then run the lisp function
(regress ‘‘foo.output’’)

If the file does not contain proper regression markups it is ignored. Comments or any other
commands in the file that are not surrounded by “—S” and “—E” boundaries are ignored.

defvar *all-tests-ran*

This variable is used to check whether all of the tests actually ran. This is needed to see if
the execution ended early.

— initvars —

(defun regress (infile)
(let (name comment test (count 0) (passed 0) (failed 0))
(declare (special *all-tests-ran*))
(setq *all-tests-ran* nil)
(with-open-file (stream infile :direction :input)
(setq name (getspoolname stream))
(when name
(format t "testing ~a~%" name)
(loop
(setq *ok* nil)
(multiple-value-setq (comment test) (findnexttest stream))
(unless comment (return))
(setq count (+ count 1))
(if (testpassed test)
(progn
(setq passed (+ passed 1))
(format t "passed ~a ~a~%" name comment))
(progn
(setq failed (+ failed 1))
(format t "FAILED ~a ~a~%" name comment)))
(if (= failed 0)
(format t "regression result passed ~a of ~a stanzas ~Tfile ~a~%"
passed count name)
(format t "regression result FAILED ~a of ~a stanzas ~Tfile ~a~%"
failed count name))
(unless *all-tests-ran*
(format t "regression result FAILED early exit in file ~a?~% name)))))

---

defun Parse test name from the spool command

We need to parse out the name of the test. The "spool" command writes a line into the output file containing the name of the test. We parse out the name of the test from this line.

— defun getspoolname 0 —

(defun getspoolname (stream)
(let (line point)
(setq line (read-line stream))
(setq point (position #\ line))
(if (or (null point)
(< (length line) 30)
(not (string= (subseq line (+ point 1) (+ point 7)) "output")))
nil
(subseq line 20 point))))

---
defun Find the next −S marker

We need to break the file into separate test cases. This routine looks for the “−S” line which indicates a test is starting. It collects up input lines until it encounters the “−E” line marking the end of the test case. These lines are returned as a list of strings.

(testnumberp p884)

— defun findnexttest —

(defun findnexttest (stream)
  (let (teststart result)
    (do ((line (read-line stream nil 'done) (read-line stream nil 'done)))
        ((or (eq line 'done) (endedp line))
         (values (if line teststart) result))
      (if teststart
       (push line result)
       (setq teststart (testnumberp line))))))

——

defun Parse out the test number from −S lines

The “−S” line has a test number on the line. We parse out the test number for printing.

(startp p887)

— defun testnumberp —

(defun testnumberp (oneline)
  (when (startp oneline) (subseq oneline 3)))

——

defvar *ok*

We can mark a test as always ok by putting the word “ok” anywhere on the start line. The regress function resets this value. The startp function checks the −S line for the word “ok”. If found, it sets this value to true which causes a failing test to be considered as passed.

—inivars —

(defvar *ok* nil "did we mark this test as always ok?")

——
defun Compare the computed and expected results

This routine takes the test input, passes it to split to clean up and break into two lists, and then compares the resulting lists element by element, complaining about any mismatches. The result is either true if everything passes or false if a mismatch occurs.

A test line can also be considered at passing if the expected line is the string “ignore”.

The ok variable allows us to mark failing tests as “ok” because we expect the test might fail due to random values or testing known bugs against expected output. We filter these tests marked “ok” so they do not count as “real” failures.

(defun testpassed (test)
  (let (answer expected (passed t) mismatchedLines)
    (declare (special *ok*))
    (multiple-value-setq (answer expected) (split test))
    (dotimes (i (length answer))
      (unless
        (or (string= (first expected) "ignore")
            (string= (first expected) (first answer)))
        (unless *ok* (setq passed nil))
        (push (cons (first expected) (first answer)) mismatchedLines))
      (pop answer)
      (pop expected))
    (when mismatchedLines
      (dolist (pair mismatchedLines)
        (format t "expected:~s~% got:~s~%" (car pair) (cdr pair))))
    passed))

---

---

defun Split the calculated and expect results into lists

We have a list containing all of the lines in a test. The input is of the form:

("--R Type: List Integer"
"--R (1) [1,4,2,-6,0,3,5,4,2,3]
"--R "
"--R "
" (1) [1,4,2,-6,0,3,5,4,2,3]
"
"l := [1,4,2,-6,0,3,5,4,2,3]"
)
It removes the “--R” prefix from the result strings and generates two hopefully equal-length lists, thus:

```
(" Type: List Integer"
 " (1) [1,4,2,- 6,0,3,5,4,2,3]"
 "")
(" Type: List Integer"
 " (1) [1,4,2,- 6,0,3,5,4,2,3]"
 "")
```

Thus the first line is the start line, the second line is the Axiom input line, followed by the Axiom output. Then we have the lines marked “--R” which are the expected result. We split these into two separate lists and throw way the lines that are the start and end lines.

Once we have classified all of the lines we need to throw away the input lines. By assumption there will be more answer lines than expected lines because the input lines are included. And given the way we process the file these input lines are on the top of the answer stack. Since the number of answer lines should equal the number of expected lines we pop the stack until the numbers are equal.

Each element of the answer list should be **string**= to the corresponding element of the result list.

If the input line starts with “--I” we push the string “ignore”. This is useful for handling random results or gensym symbols.

```
[startp p887]
[enddp p887]
[ignorep p888]
[resultp p888]

— defun split —
```

```
(defun split (test)
  (let (answer (acnt 0) expected (ecnt 0))
    (dolist (oneline test)
      (cond
        ((startp oneline))
        ((endedp oneline))
        ((ignorep oneline)
          (setq ecnt (+ ecnt 1))
          (push "ignore" expected))
        ((resultp oneline)
          (setq ecnt (+ ecnt 1))
          (push (subseq oneline 3) expected))
        (t
          (setq acnt (+ acnt 1))
          (push oneline answer))))
    (dotimes (i (- acnt ecnt)) (pop answer))
  )
```
(values (nreverse answer) (nreverse expected)))

---

defun Returns true on –S lines

This test returns true if we have a “start” line. That is, a line with a “–S” prefix.
The *all-tests-ran* variable is true if the start line is of the form ”–S N of M” and N=M, that is, it checks that all tests were performed since this should only occur on the last start line. This will detect “premature exit” in processing.
If a test is failing because of random input values or we want the test to fail but not to count toward failing values then put the string “ok” somewhere on the “–S” line as in:

```
--S 29 of 42 fails due to random values but that is ok
```

[lastcount p888]
["ok" p884]

---

defun Returns true on –E lines

This test returns true if we have a “ended” line. That is, a line with a “–E” prefix.
---

defun endedp 0

(defun endedp (oneline)
(and (>= (length oneline) 3) (string= (subseq oneline 0 3) "--E")))
defun Returns true on –R lines

This test returns true if we have a “results” line. That is, a line with a “–R” prefix.

— defun resultp 0 —

(defun resultp (oneline)
  (and (>= (length oneline) 3) (string= (subseq oneline 0 3) "--R")))

defun Returns true on –I lines

This test returns true if we have an “ignore” line. That is, a line with a “–I” prefix.

— defun ignorep 0 —

(defun ignorep (oneline)
  (and (>= (length oneline) 3) (string= (subseq oneline 0 3) "--I")))

defun Check the last –S line ran

If the “–S” line has the format “–S n of m” we return true if n=m, false otherwise. Thus,

"--S" => nil
"--S 1 of 4" => nil
"--S 10 of 40" => nil
"--S 4 of 4" => t
"--S 40 of 40" => t
"--S 1 of a" => nil

This is used as a final end check to make sure that all of the tests actually ran rather than having the regression test exit early and quietly. This will be false on all but the last test and will be false if the “–S” line does not contain the optional count marker. It is not required but is highly recommended.

— defun lastcount 0 —

(defun lastcount (oneline)
  (let ((n :done) (m :done) next somemore isof)
    (when (and (>= (length oneline) 3) (string= (subseq oneline 0 3) "--S"))
      (setq somemore (string-trim " " (subseq oneline 3)))
      (when somemore
        (multiple-value-setq (n next) (read-from-string somemore nil :done))))
(when (integerp n)
  (setq somemore (string-trim " " (subseq somemore next)))
  (multiple-value-setq (isof next) (read-from-string somemore nil :done))
  (when (string= isof "OF")
    (setq somemore (string-trim " " (subseq somemore next)))
    (multiple-value-setq (m next) (read-from-string somemore nil :done))))
(and (integerp m) (integerp n) (= m n)))

---
25.33  )savesystem Command

savesystem man page

— savesystem.help —

====================================================================
A.8. )savesystem
====================================================================

User Level Required: interpreter

Command Syntax:

- )savesystem filename

Command Description:

This command is used to save an AXIOM image to disk. This creates an
executable file which, when started, has everything loaded into it
that was there when the image was saved. Thus, after executing commands
which cause the loading of some packages, the command:

)savesystem /tmp/savesys

will create an image that can be restarted with the UNIX command:

axiom -ws /tmp/savesys

This new system will not need to reload the packages and domains that
were already loaded when the system was saved.

There is currently a restriction that only systems started with the
command "AXIOMsys" may be saved.

axiom
(1) -> t1:=4
(1) -> )savesystem foo

and Axiom exits. Then do

./foo
(1) -> t1
4

——
defvar *ThisIsARunningSystem*

When a user does

)save system foo

we set this variable to true. This is tested in the restart function, which is called when the system starts, to prevent losing user information.

— initvars —

(defvar *ThisIsARunningSystem* nil "Are we restarting a running system?")

---

defun The )save system command

[helpSpad2Cmd p813]
[spad-save p1135]

— defun savesystem —

(defun |savesystem| (arg)
  (if (or (not (eql (|#| arg) 1)) (null (symbolp (car arg)))))
    (|helpSpad2Cmd| '(|savesystem|))
  (progn
    (setq *ThisIsARunningSystem* t)
    (spad-save (symbol-name (car arg))))))

---
25.34  )set Command

    set man page

    — set.help —

====================================================================
A.21.  )set
====================================================================

User Level Required: interpreter

Command Syntax:

-  )set
-  )set label1 [... labelN]
-  )set label1 [... labelN] newValue

Command Description:

The )set command is used to view or set system variables that control what messages are displayed, the type of output desired, the status of the history facility, the way AXIOM user functions are cached, and so on. Since this collection is very large, we will not discuss them here. Rather, we will show how the facility is used. We urge you to explore the )set options to familiarize yourself with how you can modify your AXIOM working environment. There is a HyperDoc version of this same facility available from the main HyperDoc menu. Click [here] to go to it.

The )set command is command-driven with a menu display. It is tree-structured. To see all top-level nodes, issue )set by itself.

)set

Variables with values have them displayed near the right margin. Subtrees of selections have ‘‘...'’ displayed in the value field. For example, there are many kinds of messages, so issue )set message to see the choices.

)set message

The current setting for the variable that displays whether computation times are displayed is visible in the menu displayed by the last command. To see more information, issue

)set message time

This shows that time printing is on now. To turn it off, issue
As noted above, not all settings have so many qualifiers. For example, to change the )quit command to being unprotected (that is, you will not be prompted for verification), you need only issue

)set quit unprotected

Also See:
0 )quit

Overview
This section contains tree of information used to initialize the )set command in the interpreter. The current list is:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>compile</td>
<td>Library compiler options</td>
<td>...</td>
</tr>
<tr>
<td>breakmode</td>
<td>execute break processing on error break</td>
<td></td>
</tr>
<tr>
<td>expose</td>
<td>control interpreter constructor exposure</td>
<td>...</td>
</tr>
<tr>
<td>functions</td>
<td>some interpreter function options</td>
<td>...</td>
</tr>
<tr>
<td>fortran</td>
<td>view and set options for FORTRAN output</td>
<td>...</td>
</tr>
<tr>
<td>kernel</td>
<td>library functions built into the kernel for efficiency</td>
<td>...</td>
</tr>
<tr>
<td>hyperdoc</td>
<td>options in using HyperDoc</td>
<td></td>
</tr>
<tr>
<td>help</td>
<td>view and set some help options</td>
<td>...</td>
</tr>
<tr>
<td>history</td>
<td>save workspace values in a history file on</td>
<td></td>
</tr>
<tr>
<td>messages</td>
<td>show messages for various system features</td>
<td>...</td>
</tr>
<tr>
<td>naglink</td>
<td>options for NAGLink</td>
<td>...</td>
</tr>
<tr>
<td>output</td>
<td>view and set some output options</td>
<td>...</td>
</tr>
<tr>
<td>quit</td>
<td>protected or unprotected quit unprotected</td>
<td></td>
</tr>
<tr>
<td>streams</td>
<td>set some options for working with streams</td>
<td></td>
</tr>
<tr>
<td>system</td>
<td>set some system development variables</td>
<td>...</td>
</tr>
<tr>
<td>userlevel</td>
<td>operation access level of system user development</td>
<td></td>
</tr>
</tbody>
</table>

Variables with current values of ... have further sub-options. For example, issue )set system to see what the options are for system. For more information, issue )help set.

16"quit" (25.30 p 872)
Initialize the set variables

The argument settree is initially the $setOption variable. The fourth element is a union-style switch symbol. The fifth element is usually a variable to set. The sixth element is a subtree to recurse for the TREE switch. The seventh element is usually the default value. For more detailed explanations see the list structure section 25.34. [sayMSG p29]

[literals p??]
[translateYesNo2TrueFalse p899]
[tree p??]
[initializeSetVariables p894]

— defun initializeSetVariables —

(defun |initializeSetVariables| (settree)
"Initialize the set variables"
(dolist (setdata settree)
  (case (fourth setdata)
    (function
      (if (canFuncall? (fifth setdata))
        (funcall (fifth setdata) '|%initialize%|)
        (|sayMSG| (concatenate 'string " Function not implemented. "
          (package-name *package*) ":" (string (fifth setdata))))))
    (integer (set (fifth setdata) (seventh setdata)))
    (string (set (fifth setdata) (seventh setdata)))
    (literals
      (set (fifth setdata) (|translateYesNo2TrueFalse| (seventh setdata))))
    (tree (|initializeSetVariables| (sixth setdata))))))

Reset the workspace variables

[copy p??]
[initializeSetVariables p894]
[countlist p??]
[editfile p781]
[sourcefiles p??]
[pretty p??]
[spacelist p??]
[timerlist p??]
$sourceFiles p??]
$existingFiles p??]
$functionTable p770]
$boot p758]
$compileMapFlag p??]
$echoLineStack p??]
(defun resetWorkspaceVariables ()
"Reset the workspace variables"
(declare (special /countlist /editfile /sourcefiles |$sourceFiles| /pretty
 /spacelist /timerlist |$existingFiles| |$functionTable| $boot
 |$compileMapFlag| |$echoLineStack| |$operationNameList| |$slamFlag|
 |$CommandSynonymAlist| |$InitialCommandSynonymAlist|
 |$UserAbbreviationsAlist| |$msgAlist| |$msgDatabase| |$msgDatabaseName|
 |$dependeeClosureAlist| |$IOindex| |$coerceIntByMapCounter| |$e| |$env|
 |$setOptions|))
(setq /countlist nil)
(setq /editfile nil)
(setq /sourcefiles nil)
(setq |$sourceFiles| nil)
(setq /pretty nil)
(setq /spacelist nil)
(setq /timerlist nil)
(setq |$existingFiles| (make-hash-table :test #'equal))
(setq |$functionTable| nil)
(setq $boot nil)
(setq |$compileMapFlag| nil)
(setq |$echoLineStack| nil)
(setq |$operationNameList| nil)
(setq |$slamFlag| nil)
(setq |$CommandSynonymAlist| (copy |$InitialCommandSynonymAlist|))
(setq |$UserAbbreviationsAlist| nil)
(setq |$msgAlist| nil)
(setq |$msgDatabase| nil)
(setq |$msgDatabaseName| nil)
(setq |$dependeeClosureAlist| nil)
(setq $IOindex 1)
(setq |$coerceIntByMapCounter| 0)
(setq $e (cons (cons nil nil) nil))
(setq $env (cons (cons nil nil) nil))
((initializeSetVariables |$setOptions|))

Display the set option information

(defun displaySetOptionInformation (arg setdata)
  "Display the set option information"
  (let (current)
    (declare (special $linelength))
    (cond
      ((eq (fourth setdata) 'tree)
       (displaySetVariableSettings (sixth setdata) (first setdata)))
      (t
       (format t "~,,-:%T%,-%T%", (- $linelength 2)
       (concat " The " (object2String arg) " Option ")
       (sayBrightly)
       " '((%1) ,%(bright "Description:"), (second setdata)))
      (case (fourth setdata)
        (function
         (terpri)
         (if (canFuncall? (fifth setdata))
          (funcall (fifth setdata) |%describe%)
          (sayMSG " Function not implemented.")))
        (integer
         (sayMessage)
         " The" ,%(bright arg) "option" " may be followed by an integer in the range"
        )])
      ))
Display the set variable settings

(defun |displaySetVariableSettings| (settree label)
  "Display the set variable settings"
  (let (setoption opt subtree subname)
(declare (special $linelength))
(if (eq label '||)
  (setq label "")
  (setq label (concat " (" label ")")))
(format t "~v:@<~a~>~%" (- $linelength 2)
  (concat " Current Values of" label " Variables "))
(terpri)
(|sayBrightly|
  (list "Variable " "Description "
    "Current Value")
(say (|fillerSpaces| $linelength (|specialChar| 'hbar|))
(setq subtree nil)
(dolist (setdata settree)
  (when (|satisfiesUserLevel| (third setdata))
    (setq setoption (|object2String| (first setdata)))
    (setq setoption
      (concat setoption
        (|fillerSpaces| (- 13 (|#| setoption)) " ")
        (second setdata)))
    (setq setoption
      (concat setoption
        (|fillerSpaces| (- 55 (|#| setoption)) " ")
        (fourth setdata))
      (case (fourth setdata)
        (function
          (setq opt
            (if (canFuncall? (fifth setdata))
              (funcall (fifth setdata) '|%display%|
              "unimplemented"))
            (cond
              ((cons opt)
                (setq opt
                  (do ((t2 opt (cdr t2)) t1 (o nil))
                      ((or (atom t2) (progn (setq o (car t2)) nil)) t1)
                    (setq t1 (append t1 (cons o (cons " 
nil))))))
              (|sayBrightly| (|concat| setoption opt)))
        (string
          (setq opt (|object2String| (|eval| (fifth setdata))))
          (|sayBrightly| '(,setoption ,@(|bright| opt))))
        (integer
          (setq opt (|object2String| (|eval| (fifth setdata))))
          (|sayBrightly| '(,setoption ,@(|bright| opt))))
        (literals
          (setq opt (|object2String| (|translateTrueFalse2YesNo| (|eval| (fifth setdata))))
          (|sayBrightly| '(,setoption ,@(|bright| opt))))
        (TREE
          (|sayBrightly| '(,setoption ,@(|bright| "...")))
          (setq subtree t)
          (setq subname (|object2String| (first setdata)))))))
(terpri)
(when subtree
  (sayBrightly|
    '("Variables with current values of" ,@(|bright| "...")
    "have further sub-options. For example,"))
  (sayBrightly|
    '("issue" ,@(|bright| "set ") ,subname
    " to see what the options are for" ,@(|bright| subname) "."
    |%l| "For more information, issue" ,@(|bright| "help set") ".")))

---

Translate options values to t or nil

(member p1198)

— defun translateYesNo2TrueFalse —

(defun translateYesNo2TrueFalse (x)
  "Translate options values to t or nil"
  (cond
   ((member x '(|yes| |on|)) t)
   ((member x '(|no| |off|)) nil)
   (t x)))

---

Translate t or nil to option values

— defun translateTrueFalse2YesNo —

(defun translateTrueFalse2YesNo (x)
  "Translate t or nil to option values"
  (cond
   ((eq x t) '|on|)
   ((null x) '|off|)
   (t x)))

---

The list structure

The structure of each list item consists of 7 items. Consider this example:
(userlevel
 "operation access level of system user"
 interpreter
 LITERALS
 $UserLevel
 (interpreter compiler development)
 development)

The list contains (the names in bold are accessor names that can be found in property.lisp.pamphlet. Look for "setName"):  

1 Name the keyword the user will see. In this example the user would say "\)set output userlevel".

2 Label the message the user will see. In this example the user would see "operation access level of system user".

3 Level the level where the command will be accepted. There are three levels: interpreter, compiler, development. These commands are restricted to keep the user from causing damage.

4 Type a symbol, one of FUNCTION, INTEGER, STRING, LITERALS, FILENAME or TREE.

5 Var

FUNCTION is the function to call

INTEGER is the variable holding the current user setting.

STRING is the variable holding the current user setting.

LITERALS variable which holds the current user setting.

FILENAME is the variable that holds the current user setting.

TREE

6 Leaf

FUNCTION is the list of all possible values

INTEGER is the range of possible values

STRING is a list of all possible values

LITERALS is a list of all of the possible values

FILENAME is the function to check the filename

TREE

7 Def is the default value

FUNCTION is the default setting

INTEGER is the default setting
25.35. **SET BREAKMODE**

STRING is the default setting
LITERALS is the default setting
FILENAME is the default value
TREE

### 25.35 set breakmode

--------------------- The breakmode Option ---------------------

Description: execute break processing on error

The breakmode option may be followed by any one of the following:

- nobreak
- -> break
- query
- resume
- fastlinks
- quit

The current setting is indicated.

```lisp
(defvar $BreakMode |initvars|
  (defvar |$BreakMode| 'nobreak "execute break processing on error")

------

--- breakmode ---

(|breakmode|
  "execute break processing on error"
  |interpreter|
  LITERALS
  |$BreakMode|
  (|nobreak| |break| |query| |resume| |fastlinks| |quit|)
  |nobreak|) ; needed to avoid possible startup looping

------
```
25.36 set debug

Current Values of debug Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lambdatype</td>
<td>Show type information for #1 syntax</td>
<td>off</td>
</tr>
<tr>
<td>dalymode</td>
<td>Interpret leading open paren as lisp</td>
<td>off</td>
</tr>
</tbody>
</table>

---

The lambdatype Option

Description: Show type information for #1 syntax

defvar $lambdatype

-- initvars --

(defvar $lambdatype nil "show type information for #1 syntax")

---

-- debuglambdatype --

(||lambdatype||
 "show type information for #1 syntax")
25.37.  SET COMPILER

<table>
<thead>
<tr>
<th>current Values of compiler Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>output</td>
</tr>
<tr>
<td>input</td>
</tr>
</tbody>
</table>

---

25.37  set compiler

---

set compiler output

---------------------- The output Option ----------------------

Description: library in which to place compiled code

---

set compiler output

-----------

"library in which to place compiled code "

CHAPTER 25. SYSTEM COMMAND HANDLING

The set output command handler

(defun setOutputLibrary (arg)
  "The set output command handler"
  (let (fn)
    (declare (special $outputLibraryName))
    (cond
      ((eq arg '|%initialize%|) (setq $outputLibraryName nil))
      ((eq arg '|%display%|) (or $outputLibraryName "user.lib"))
      ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?) (/= (|#| arg) 1))
        (describeOutputLibraryArgs))
      (t
       (when (probe-file (setq fn (princ-to-string (car arg))))
         (setq fn (truename fn)))
       (openOutputLibrary (setq $outputLibraryName fn)))))

Describe the set output library arguments

(defun describeOutputLibraryArgs ()
  "Describe the set output library arguments"
  (sayBrightly (list
    "set compile output library is used to tell the compiler where to place"
    '|%l| "compiled code generated by the library compiler. By default it goes"
    '|%l| "in a file called user.lib in the current directory.")))
defvar output-library

--- initvars ---

(defvar output-library nil)

--- Open the output library ---

The input-libraries and output-library are now truename based.

[dropInputLibrary p908]
[output-library p905]
[input-libraries p907]

--- defun openOutputLibrary ---

(defun openOutputLibrary (lib)
"Open the output library"
(defun (special output-library input-libraries))
(dropInputLibrary lib)
(setq output-library (truename lib))
(push output-library input-libraries))

--- set compiler input ---

------------------ The input Option ------------------

Description: controls libraries from which to load compiled code

)set compile input add library is used to tell AXIOM to add
library to the front of the path which determines where
compiled code is loaded from.
)set compile input drop library is used to tell AXIOM to remove
library from this path.
— compileinput —

\[
\begin{align*}
| \text{input} & | \\
& "\text{controls libraries from which to load compiled code}" \text{ } \\
| \text{interpreter} & | \\
& \text{FUNCTION} \text{ } \\
| \text{setInputLibrary} & | \\
& \text{NIL} \text{ } \\
| \text{htSetInputLibrary} & | \\
\end{align*}
\]

The set input library command handler

The input-libraries is now maintained as a list of truenames.

\[
\begin{align*}
| \text{describeInputLibraryArgs p} & | \\
| \text{qcar p} & | \\
| \text{qcdr p} & | \\
| \text{selectOptionLC p} & | \\
| \text{addInputLibrary p} & | \\
| \text{dropInputLibrary p} & | \\
| \text{input-libraries p} & | \\
\end{align*}
\]

— defun setInputLibrary —

\[
\begin{align*}
| \text{defun setInputLibrary |} \\
& (arg) \text{ } \\
& "\text{The set input library command handler}" \text{ } \\
& (\text{declare } (\text{special } \text{input-libraries})) \text{ } \\
& (\text{let } (\text{tmp1 filename act})) \text{ } \\
& (\text{cond }) \text{ } \\
& \quad ((\text{eq arg } '|\%initialize%|) \text{ t}) \text{ } \\
& \quad ((\text{eq arg } '|\%display%|) (\text{mapcar } '#\text{namestring } \text{input-libraries})) \text{ } \\
& \quad ((\text{or } (\text{null arg}) (\text{eq arg } '|\%describe%|) (\text{eq (car arg) } '?)) \text{ } \\
& \quad ((\text{describeInputLibraryArgs})) \text{ } \\
& \quad ((\text{and } (\text{consp arg})) \text{ } \\
& \quad \quad (\text{progn}) \text{ } \\
& \quad \quad (\text{setq act (qcar arg)}) \text{ } \\
& \quad \quad (\text{setq tmp1 (qcdr arg)}) \text{ } \\
& \quad \quad (\text{and } (\text{consp tmp1})) \text{ } \\
& \quad \quad \quad (\text{eq (qcdr tmp1) nil}) \text{ } \\
& \quad \quad \quad (\text{progn (setq filename (qcar tmp1)) t})) \text{ } \\
& \quad \quad \quad (\text{setq act } (\text{selectOptionLC} \text{ act '|(add| drop|) nil})) \text{ } \\
& \quad \quad (\text{cond}) \text{ } \\
& \quad \quad \quad ((\text{eq act '|add|}) \text{ } \\
& \quad \quad \quad \quad (\text{addInputLibrary} \text{ (truename (princ-to-string filename))}))
\end{align*}
\]
((eq act 'drop)
 (dropInputLibrary (truename (princ-to-string filename))))
(t (setInputLibrary nil))))

Describe the set input library arguments
[sayBrightly p??]

— defun describeInputLibraryArgs —

(defun describeInputLibraryArgs ()
"Describe the set input library arguments"
(sayBrightly (list
 " )set compile input add library "
"is used to tell AXIOM to add library to"
'|%l| " the front of the path used to find compile code."
'|%l|
" )set compile input drop library is used to tell AXIOM to remove library"
'|%l| " from this path.")))

Add the input library to the list

The input-libraries variable is now maintained as a list of truenames. [dropInputLibrary p908]
[input-libraries p907]

— defun addInputLibrary —

(defun addInputLibrary (lib)
"Add the input library to the list"
(declare (special input-libraries))
(dropInputLibrary lib)
(push (truename lib) input-libraries))

defvar input-libraries

— initvars —
(defvar input-libraries nil)

---

Drop an input library from the list

[input-libraries p907]

— defun dropInputLibrary —

(defun dropInputLibrary (lib)
  "Drop an input library from the list"
  (declare (special input-libraries))
  (setq input-libraries (delete (truename lib) input-libraries :test #'equal)))

25.38 set debug dalymode

The $dalymode$ variable is used in a case statement in intloopReadConsole. This variable can be set to any non-nil value. When not nil the interpreter will send any line that begins with an "(" to be sent to the underlying lisp. This is useful for debugging Axiom. The normal value of this variable is NIL.

This variable was created as an alternative to prefixing every lisp command with )lisp. When doing a lot of debugging this is tedious and error prone. This variable was created to shortcut that process. Clearly it breaks some semantics of the language accepted by the interpreter as parens are used for grouping expressions.

---------------------- The dalymode Option ----------------------

Description: Interpret leading open paren as lisp

defvar dalymode

— initvars —

(defvar $dalymode nil "Interpret leading open paren as lisp")
25.39. **SET EXPOSE**

--- debugdalymode ---

```
(dalymode)
"Interpret leading open paren as lisp"
(interpreter)
LITERALS
$dalymode
(|on| |off|)
|off|)
```

---

### 25.39 set expose

---------------------- The expose Option ----------------------

**Description:** control interpreter constructor exposure

The following groups are explicitly exposed in the current frame (called initial):

```
  basic
  categories
  naglink
  anna
```

The following constructors are explicitly exposed in the current frame:

```
  there are no explicitly exposed constructors
```

The following constructors are explicitly hidden in the current frame:

```
  there are no explicitly hidden constructors
```

When `)set expose` is followed by no arguments, the information you now see is displayed. When followed by the `initialize` argument, the exposure group data in the file `interp.exposed` is read and is then available. The arguments `add` and `drop` are used to add or drop exposure groups or explicit constructors from the local frame exposure data. Issue

```
)set expose add  or  )set expose drop
```

for more information.

--- expose ---

```
(expose)
```
"control interpreter constructor exposure"
|interpreter|
FUNCTION
|setExpose|
NIL
|htSetExpose|

functions

Current Values of functions Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache</td>
<td>number of function results to cache</td>
<td>0</td>
</tr>
<tr>
<td>compile</td>
<td>compile, don't just define function bodies off</td>
<td></td>
</tr>
<tr>
<td>recurrence</td>
<td>specially compile recurrence relations</td>
<td>on</td>
</tr>
</tbody>
</table>

functions cache

---------------------- The cache Option ----------------------

Description: number of function results to cache

)set functions cache is used to tell AXIOM how many values computed by interpreter functions should be saved. This can save quite a bit of time in recursive functions, though one must consider that the cached values will take up (perhaps
valuable) room in the workspace.

The value given after cache must either be the word all or a positive integer. This may be followed by any number of function names whose cache sizes you wish to so set. If no functions are given, the default cache size is set.

Examples:  )set fun cache all
             )set fun cache 10 f g Legendre

In general, functions will cache no returned values.

--- functionscache ---

(|cache| "number of function results to cache"
|interpreter|
FUNCTION |setFunctionsCache|
NIL |htSetCache|)

---

defvar $cacheAlist

--- initvars ---

(defvar |$cacheAlist| nil)

---

The top level set functions cache handler

[object2String p??]
[describeSetFunctionsCache p914]
[sayAllCacheCounts p914]
[sayMessage p??]
[bright p??]
[terminateSystemCommand p724]
[countCache p912]
[$options p??]
[$cacheCount p??]
(defun |setFunctionsCache| (arg)
  "The top level set functions cache handler"
  (let (|#options| n)
    (declare (special |#options| |#cacheCount| |#cacheAlist|))
    (cond
      ((eq arg '|%initialize%|)
        (setq |#cacheCount| 0)
        (setq |#cacheAlist| nil))
      ((eq arg '|%display%|)
        (if (null |#cacheAlist|)
            (|object2string| |#cacheCount|)
            "..."))
      ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
        (|describeSetFunctionsCache|
          (terpri)
          (|sayAllCacheCounts|))
        t
        (setq n (car arg))
        (cond
          ((and (not (eq n '|all|)) (or (null (integerp n)) (minusp n)))
            (|sayMessage|
              `(Your value of ,@(|bright| n) "is invalid because ..."))
            (|describeSetFunctionsCache|
              (|terminateSystemCommand|))
            (t
              (when (cdr arg) (list (cons '|vars| (cdr arg))))
              (|countCache| n)))))))

----

Display a particular cache count

[qcdr p??]
[qcar p??]
[identp p1197]
[sayKeyedMsg p27]
[insertAlist p913]
[internl p??]
[sayCacheCount p915]
[optionError p721]
[#options p??]
[#cacheAlist p911]
[#cacheCount p??]
— defun countCache —

(defun countCache (n)
"Display a particular cache count"
(let (tmp1 l cachecountname)
  (declare (special |$options| |$cacheAlist| |$cacheCount|))
  (cond
    (!|$options|
      (cond
        ((and (consp |$options|)
          (eq (qcdr |$options|)) nil)
          (progn
            (setq tmp1 (qcar |$options|))
            (and (consp tmp1)
              (eq (qcar tmp1) '|vars|)
              (progn (setq l (qcdr tmp1)) t))))
        (t (|optionError| (caar |$options|) nil))))
    (dolist (x l)
      (if (null (identp x))
        (|sayKeyedMsg| "%1 is not a valid function name." (list x))
        (progn
          (setq |$cacheAlist| (|insertAlist| x n |$cacheAlist|))
          (setq cachecountname (internl x ";COUNT")
          (set cachecountname n)
          (|sayCacheCount| x n))))
    (t (|optionError| (caar |$options|) nil)))))

defun insertAlist

[rplac p??]
[?order p??]
— defun insertAlist —

(defun insertAlist (a b z)
  (labels ((fn (a b z)
         (cond
           ((null (cdr z)) (rplac (cdr z) (list (cons a b))))
           ((equal a (elt (elt z 1) 0)) (rplac (cdr (elt z 1)) b))
           ((?order (elt (elt z 1) 0) a) (rplac (cdr z) (cons (cons a b) (cdr z))))
           (t (fn a b (cdr z))))))
    (cond
      (null z) (fn a b z)
      (t (fn a b (cdr z))))))
((null z) (list (cons a b)))
((equal a (elt (elt z 0) 0)) (rplac (cdr z) b) z)
((?order (elt (elt z 0) 0) a) (cons (cons a b) z))
(t (fn a b z)))

Describe the set functions cache

| sayBrightly p? |

— defun describeSetFunctionsCache —

(defun |describeSetFunctionsCache| ()
"Describe the set functions cache"
|sayBrightly| (list
"set functions cache"
"is used to tell AXIOM how many"
'|%1| " values computed by interpreter functions should be saved. This"
'|%1| " can save quite a bit of time in recursive functions, though one"
'|%1| " must consider that the cached values will take up (perhaps"
'|%1| " valuable) room in the workspace."
'|%1| 
'|%1| " The value given after "
"cache must either be the word all or a positive integer."
'|%1| " This may be followed by any number of function names whose cache"
'|%1| " sizes you wish to so set. If no functions are given, the default"
'|%1| " cache size is set."
'|%1| 
'|%1| " Examples:"
'|%1| " )set fun cache all )set fun cache 10 f g Legendre")

Display all cache counts

| sayCacheCount p915 |
| $cacheCount p? |
| $cacheAlist p911 |

— defun sayAllCacheCounts —

(defun |sayAllCacheCounts| ()
"Display all cache counts"
(let (x n)
(declare (special |$cacheCount| |$cacheAlist|))
(|sayCacheCount| nil |$cacheCount|)
(when |$cacheAlist|
  (do ((t0 |$cacheAlist| (cdr t0)) (t1 nil))
      ((or (atom t0)
           (progn (setq t1 (car t0)) nil)
           (progn
              (progn
                (setq x (car t1)) (setq n (cdr t1)) t1)
              nil))
       nil)
  (when (not (equal n |$cacheCount|)) (|sayCacheCount| x n)))))

---

Describe the cache counts

[bright p??]
[linearFormatName p??]
[sayBrightly p??]

— defun sayCacheCount —

(defun |sayCacheCount| (fn n)
"Describe the cache counts"
(let (prefix phrase)
  (setq prefix
    (cond
      (fn (cons 'function (|bright| (|linearFormatName| fn))))
      ((eql n 0) (list '|interpreter functions |))
      (t (list '|In general, interpreter functions |)))
    (cond
      ((eql n 0)
       (cond
        (fn
         (|sayBrightly|
          '(" Caching for " ,prefix "is turned off")))
        t
         (|sayBrightly| " In general, functions will cache no returned values.")))
      (t
       (setq phrase
         (cond
          ((eq n '|all|) `(|values.|))
          ((eql n 1) (list '| only the last value.|))
          (t `(| the last| ,@(|bright| n) |values.|)))
          (|sayBrightly| '(" " ,@prefix "will cache" ,@phrase)))))))
functions compile

--------------------- The compile Option ---------------------

Description: compile, don’t just define function bodies

The compile option may be followed by any one of the following:

-> on
    off

The current setting is indicated.

\defdollar{compileDontDefineFunctions}
\begin{chunk}{initvars}
(defvar |$compileDontDefineFunctions| t
  "compile, don’t just define function bodies")
\end{chunk}
\begin{chunk}{functionscompile}
(|compile|
  "compile, don’t just define function bodies"
  |interpreter|
  LITERALS
  |$compileDontDefineFunctions|
  (|on| |off|)
  |on|)
\end{chunk}
\subsection{functions recurrence}
\begin{verbatim}
-------------------- The recurrence Option --------------------

Description: specially compile recurrence relations

The recurrence option may be followed by any one of the following:

-> on
    off

The current setting is indicated.
defvar $compileRecurrence

— initvars —

(defvar $compileRecurrence t "specially compile recurrence relations")

— functionsrecurrence —

{|recurrence|
"specially compile recurrence relations"
|interpreter|
LITERALS
|$compileRecurrence|
(|on| |off|)
|on|

— fortran —

25.40 set fortran

Current Values of fortran Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ints2floats</td>
<td>where sensible, coerce integers to reals</td>
<td>on</td>
</tr>
<tr>
<td>fortindent</td>
<td>the number of characters indented</td>
<td>6</td>
</tr>
<tr>
<td>fortlength</td>
<td>the number of characters on a line</td>
<td>72</td>
</tr>
<tr>
<td>typedecs</td>
<td>print type and dimension lines</td>
<td>on</td>
</tr>
<tr>
<td>defaulttype</td>
<td>default generic type for FORTRAN object</td>
<td>REAL</td>
</tr>
<tr>
<td>precision</td>
<td>precision of generated FORTRAN objects</td>
<td>double</td>
</tr>
<tr>
<td>intrinsic</td>
<td>whether to use INTRINSIC FORTRAN functions</td>
<td>off</td>
</tr>
<tr>
<td>exmlength</td>
<td>character limit for FORTRAN expressions</td>
<td>1320</td>
</tr>
<tr>
<td>segment</td>
<td>split long FORTRAN expressions</td>
<td>on</td>
</tr>
<tr>
<td>optlevel</td>
<td>FORTRAN optimisation level</td>
<td>0</td>
</tr>
<tr>
<td>startindex</td>
<td>starting index for FORTRAN arrays</td>
<td>1</td>
</tr>
<tr>
<td>calling</td>
<td>options for external FORTRAN calls</td>
<td>...</td>
</tr>
</tbody>
</table>

Variables with current values of ... have further sub-options. For example, issue )set calling to see what the options are for calling. For more information, issue )help set.
(\texttt{fortran} \\
"view and set options for FORTRAN output" \\
|interpreter| \\
TREE \\
|novar| \\
( \\
\getchunk{fortranints2floats} \\
\getchunk{fortranfortindent} \\
\getchunk{fortranfortlength} \\
\getchunk{fortrantypedecs} \\
\getchunk{fortrandondefaulttype} \\
\getchunk{fortranprecision} \\
\getchunk{fortranintrinsic} \\
\getchunk{fortranexplelength} \\
\getchunk{fortransement} \\
\getchunk{fortranoptlevel} \\
\getchunk{fortranstartindex} \\
\getchunk{fortrancalling} \\
))

\textbf{set ints2floats}

\textbf{--------------------- The ints2floats Option ---------------------}

\textbf{Description: where sensible, coerce integers to reals}

The ints2floats option may be followed by any one of the following:

\textbf{-> on}

\textbf{off}

The current setting is indicated.

\textbf{defvar $fortInts2Floats}

\textbf{--- initvars ---}

(defvar |$fortInts2Floats| t "where sensible, coerce integers to reals")
— fortranInts2floats —

(ints2floats)
"where sensible, coerce integers to reals"
(interpreter)
LITERALS
$fortints2floats
(on | off)
on

set fortindent

------------------- The fortindent Option -------------------

Description: the number of characters indented

The fortindent option may be followed by an integer in the range 0 to inclusive. The current setting is 6

defvar $fortIndent

— initvars —

(defvar $fortIndent 6 "the number of characters indented")

———

— fortranfortindent —

(fortindent)
"the number of characters indented"
(interpreter)
INTEGER
$fortIndent
(0 NIL)
6)
set fortlength

------------------ The fortlength Option ------------------

Description: the number of characters on a line

The fortlength option may be followed by an integer in the range
1 to inclusive. The current setting is 72

defvar $fortLength

--- initvars ---

(defvar $fortLength 72 "the number of characters on a line")

---

--- fortranfortlength ---

([fortlength]
"the number of characters on a line"
[interpreter]
INTERGER
[$fortLength]
(1 NIL)
72)

---

settypedecs

---------------- The typedecs Option ----------------

Description: print type and dimension lines

The typedecs option may be followed by any one of the following:

-> on
  off

The current setting is indicated.
defvar $printFortranDecs

—— initvars ———

(defvar |$printFortranDecs| t "print type and dimension lines")

————-

— fortrantypedecsn —

(|typedecs|
 "print type and dimension lines"
 |interpreter|
 LITERALS
 |$printFortranDecs|
 (|on| |off|)
 |on|)

————-

set defaulttype

------------------- The defaulttype Option -------------------

Description: default generic type for FORTRAN object

The defaulttype option may be followed by any one of the following:

- REAL
- INTEGER
- COMPLEX
- LOGICAL
- CHARACTER

The current setting is indicated.

defvar $defaultFortranType

—— initvars ———

(defvar |$defaultFortranType| 'real "default generic type for FORTRAN object")
CHAPTER 25. SYSTEM COMMAND HANDLING

---

—fortrandefaulttype—

(|defaulttype|
"default generic type for FORTRAN object"
|interpreter|
LITERALS
|$defaultFortranType|
(REAL INTEGER COMPLEX LOGICAL CHARACTER)
REAL)

---

set precision

------------------------ The precision Option ------------------------

Description: precision of generated FORTRAN objects

The precision option may be followed by any one of the following:

   single
   \rightarrow double

The current setting is indicated.

defvar $fortranPrecision

---

—fortranprecision—

(|precision|
"precision of generated FORTRAN objects"
|interpreter|
LITERALS
|$fortranPrecision|
25.40. SET FORTRAN

(set intrinsic)

------------- The intrinsic Option -------------

Description: whether to use INTRINSIC FORTRAN functions

The intrinsic option may be followed by any one of the

following:

  on
  -> off

The current setting is indicated.

(defvar $useIntrinsicFunctions)

— initvars —

(defvar $useIntrinsicFunctions nil
  "whether to use INTRINSIC FORTRAN functions")

— fortranintrinsic —

{[intrinsic]
  "whether to use INTRINSIC FORTRAN functions"
  [interpreter]
  LITERALS
  [$useIntrinsicFunctions]
  {[on] [off]}
  [off]}

set explength

---------------- The explength Option ----------------
CHAPTER 25. SYSTEM COMMAND HANDLING

Description: character limit for FORTRAN expressions

The explength option may be followed by an integer in the range 0 to inclusive. The current setting is 1320

**defvar $maximumFortranExpressionLength**

--- initvars ---

(defvar $maximumFortranExpressionLength 1320
"character limit for FORTRAN expressions")

--- fortranexplength ---

(|explength|
"character limit for FORTRAN expressions" |interpreter|
INTEGER |$maximumFortranExpressionLength| (0 NIL) 1320)

---

**set segment**

---------------- The segment Option ----------------

Description: split long FORTRAN expressions

The segment option may be followed by any one of the following:

-> on
    off

The current setting is indicated.
defvar $fortranSegment

— initvars —

(defvar $fortranSegment t "split long FORTRAN expressions")

———

— fortransegment —

(defvar $fortranSegment t "split long FORTRAN expressions")

set optlevel

--------------------- The optlevel Option ---------------------

Description: FORTRAN optimisation level

The optlevel option may be followed by an integer in the range 0 to 2 inclusive. The current setting is 0

defvar $fortranOptimizationLevel

— initvars —

(defvar $fortranOptimizationLevel 0 "FORTRAN optimisation level")

———

— fortranoptlevel —

(defvar $fortranOptimizationLevel 0 "FORTRAN optimisation level")
CHAPTER 25. SYSTEM COMMAND HANDLING

set startindex

------------------- The startindex Option -------------------

Description: starting index for FORTRAN arrays

The startindex option may be followed by an integer in the range 0 to 1 inclusive. The current setting is 1

defvar $fortranArrayStartingIndex

|initvars|
(defvar $fortranArrayStartingIndex 1 "starting index for FORTRAN arrays")

---

--- fortranstartindex ---

(|startindex|
 "starting index for FORTRAN arrays"
|interpreter|
INTEGER
|$fortranArrayStartingIndex|
(0 1)
1)

---

set calling

Current Values of calling Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
</table>


tempfile set location of temporary data files /tmp/
directory set location of generated FORTRAN files ./
linker linker arguments (e.g. libraries to search) -lxlf

--- fortrancalling ---

(set fortran calling tempfile)
(set fortran calling directory)
(set fortran calling linker)

---

set tempfile

--------------------- The tempfile Option ---------------------

Description: set location of temporary data files

)set fortran calling tempfile is used to tell AXIOM where
to place intermediate FORTRAN data files. This must be the
name of a valid existing directory to which you have permission
to write (including the final slash).

Syntax:
  )set fortran calling tempfile DIRECTORYNAME

The current setting is /tmp/

defvar $fortranTmpDir

--- initvars ---

(defvar |$fortranTmpDir| "/tmp/" "set location of temporary data files ")
The top level set fortran calling tempfile handler

(defun setFortTmpDir (arg)
  "The top level set fortran calling tempfile handler"
  (let (mode)
    (declare (special $fortranTmpDir))
    (cond
      ((eq arg '|%initialize%|) (setq $fortranTmpDir "|%initialize%|")
       (eq arg '|%display%|)
       (if (stringp $fortranTmpDir)
           $fortranTmpDir
           (pname $fortranTmpDir)))
      ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
       (|describeSetFortTmpDir|))
      ((null (setq mode (|validateOutputDirectory| arg)))
       (|sayBrightly|
        " Sorry, but your argument(s)" ,0(|bright| arg)
        " is(are) not valid." |%l|)
       (|describeSetFortTmpDir|))
      (t (setq $fortranTmpDir mode)))))
Validate the output directory

— defun validateOutputDirectory —

(defun validateOutputDirectory (x)
"Validate the output directory"
(let ((dirname (car x)))
  (when (and (pathname-directory dirname) (null (probe-file dirname)))
    dirname)))

Describe the set fortran calling tempfile

[sayBrightly p??]
[$fortranTmpDir p927]

— defun describeSetFortTmpDir —

(defun describeSetFortTmpDir ()
"Describe the set fortran calling tempfile"
(declare (special $fortranTmpDir))
(sayBrightly (list
  " )set fortran calling tempfile"
  " is used to tell AXIOM where"
  " to place intermediate FORTRAN data files. This must be the "
  " name of a valid existing directory to which you have permission "
  " to write (including the final slash)."
  " "
  " Syntax:
  " )set fortran calling tempfile DIRECTORYNAME"
  " "
  " The current setting is "
  $fortranTmpDir) )

directory

------------------ The directory Option ------------------
Description: set location of generated FORTRAN files

)set fortran calling directory is used to tell AXIOM where
to place generated FORTRAN files. This must be the name
of a valid existing directory to which you have permission
to write (including the final slash).

Syntax:
)set fortran calling directory DIRECTORYNAME

The current setting is ./

defvar $fortranDirectory

— initvars —

(defvar |$fortranDirectory| "./" "set location of generated FORTRAN files ")

— callingdirectory —

{|directory|
"set location of generated FORTRAN files "
|interpreter|
FUNCTION
|setFortDir|
(""enter directory name for which you have write-permission "
DIRECTORY
|$fortranDirectory|
\chkDirectory\|"./")
NIL)

—

defun setFortDir

[pname p1195]
[describeSetFortDir p931]
[validateOutputDirectory p929]
[sayBrightly p??]
[bright p??]
--- defun setFortDir ---

(defun setFortDir (arg)
  (declare (special $fortranDirectory)))
(let (mode)
  (COND
    ((eq arg ’%initialize%) (setq $fortranDirectory “./”))
    ((eq arg ’%display%) (setq $fortranDirectory)
     (if (stringp $fortranDirectory)
         (pname $fortranDirectory)))
    ((or (null arg) (eq arg ’%describe%) (eq (car arg) ’?))
     (describeSetFortDir))
    ((null (setq mode (validateOutputDirectory arg)))
     (sayBrightly
      “(” Sorry, but your argument(s)” ,@(|bright| arg)
      “is(are) not valid.” |%l|))
     (describeSetFortDir))
    (t (setq $fortranDirectory mode)))))

---

defun describeSetFortDir

[sayBrightly p??]

--- defun describeSetFortDir ---

(defun describeSetFortDir ()
  (declare (special $fortranDirectory))
  (sayBrightly (list
    "set fortran calling directory"
    "is used to tell AXIOM where"
    "to place generated FORTRAN files. This must be the name"
    "of a valid existing directory to which you have permission"
    "to write (including the final slash)."
    "Syntax:"
    "set fortran calling directory DIRECTORYNAME"
    "The current setting is"
    $fortranDirectory))
linker

----------------------------- The linker Option -----------------------------

Description: linker arguments (e.g. libraries to search)

)set fortran calling linkerargs is used to pass arguments to the linker when using mkFort to create functions which call Fortran code. For example, it might give a list of libraries to be searched, and their locations. The string is passed verbatim, so must be the correct syntax for the particular linker being used.

Example: )set fortran calling linker "-lxlf"

The current setting is -lxlf

defvar $fortranLibraries

— initvars —

(defvar $fortranLibraries |"-lxlf"
  "linker arguments (e.g. libraries to search)"
)

— callinglinker —

{|linker|
  "linker arguments (e.g. libraries to search)"
|interpreter|
  FUNCTION
|setLinkerArgs|
  ("enter linker arguments "
  STRING
  $fortranLibraries|
  |chkDirectory|
  "-lxlf")
  NIL
  )

|
25.40. SET FORTRAN

defun setLinkerArgs

(defun setLinkerArgs (arg)
  (declare (special $fortranLibraries))
  (cond
    ((eq arg '|%initialize%|) (setq $fortranLibraries "-lxlf"))
    ((eq arg '|%display%|) (object2String $fortranLibraries))
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
      (describeSetLinkerArgs))
    ((and (listp arg) (stringp (car arg)))
      (setq $fortranLibraries (car arg)))
    (t (describeSetLinkerArgs)))))

defun describeSetLinkerArgs

(defun describeSetLinkerArgs ()
  (declare (special $fortranLibraries))
  (sayBrightly (list
    " set fortran calling linkerargs"
    " is used to pass arguments to the linker"
    " when using "
    "mkFort to create functions which call Fortran code."
    " For example, it might give a list of libraries to be searched,"
    " and their locations."
    " The string is passed verbatim, so must be the correct syntax for"
    " the particular linker being used."
    " Example: )set fortran calling linker "\"-lxlf\""
    " The current setting is "
    $fortranLibraries)))

________
25.41 set hyperdoc

Current Values of hyperdoc Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullscreen</td>
<td>use full screen for this facility</td>
<td>off</td>
</tr>
<tr>
<td>mathwidth</td>
<td>screen width for history output</td>
<td>120</td>
</tr>
</tbody>
</table>

— hyperdoc —

\{|hyperdoc| "options in using HyperDoc" \}

| interpreter |
| TREE        |
| novar       |

{|fullscreen| \getchunk{hyperdocfullscreen}
{\getchunk{hyperdocmathwidth}}
}

fullscreen

-------------- The fullscreen Option --------------

Description: use full screen for this facility

The fullscreen option may be followed by any one of the following:

on
-> off

The current setting is indicated.

defvar $fullScreenSysVars

— initvars —

(defvar |$fullScreenSysVars| nil "use full screen for this facility")
25.41. SET HYPERDOC

———

— hyperdocfullscreen —

|fullscreen|
"use full screen for this facility"
|interpreter|
LITERALS
|$fullScreenSysVars|
(|on| |off|)
|off|)

———

mathwidth

---------------------- The mathwidth Option ----------------------

Description: screen width for history output

The mathwidth option may be followed by an integer in the range 0 to inclusive. The current setting is 120

defvar $historyDisplayWidth

—— initvars —

(defvar |$historyDisplayWidth| 120 "screen width for history output")

———

— hyperdocmathwidth —

|mathwidth|
"screen width for history output"
|interpreter|
INTEGER
|$historyDisplayWidth|
((0 NIL)
120)

———
25.42 set help

Current Values of help Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullscreen</td>
<td>use fullscreen facility, if possible</td>
<td>on</td>
</tr>
</tbody>
</table>

---

fullscreen

-------------------- The fullscreen Option -------------------

Description: use fullscreen facility, if possible

The fullscreen option may be followed by any one of the following:

- on
- off

The current setting is indicated.

defvar $useFullScreenHelp

--- initvars ---

(defvar $useFullScreenHelp t "use fullscreen facility, if possible")
25.43. SET HISTORY

--------------------- The history Option ---------------------

Description: save workspace values in a history file

The history option may be followed by any one of the
following:

-> on
   off

The current setting is indicated.

defvar $HiFiAccess

--- initvars ---

(defvar |$HiFiAccess| t "save workspace values in a history file")

---

--- history ---

(|history|
 "save workspace values in a history file"
 |interpreter|
 LITERALS)
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoload</td>
<td>print file auto-load messages</td>
<td>off</td>
</tr>
<tr>
<td>bottomup</td>
<td>display bottom up modemap selection</td>
<td>off</td>
</tr>
<tr>
<td>coercion</td>
<td>display datatype coercion messages</td>
<td>off</td>
</tr>
<tr>
<td>dropmap</td>
<td>display old map defn when replaced</td>
<td>off</td>
</tr>
<tr>
<td>expose</td>
<td>warning for unexposed functions</td>
<td>off</td>
</tr>
<tr>
<td>file</td>
<td>print msgs also to SPADMSG LISTING</td>
<td>off</td>
</tr>
<tr>
<td>frame</td>
<td>display messages about frames</td>
<td>off</td>
</tr>
<tr>
<td>highlighting</td>
<td>use highlighting in system messages</td>
<td>off</td>
</tr>
<tr>
<td>instant</td>
<td>present instantiation summary</td>
<td>off</td>
</tr>
<tr>
<td>insteach</td>
<td>present instantiation info</td>
<td>off</td>
</tr>
<tr>
<td>interponly</td>
<td>say when function code is interpreted</td>
<td>on</td>
</tr>
<tr>
<td>number</td>
<td>display message number with message</td>
<td>off</td>
</tr>
<tr>
<td>prompt</td>
<td>set type of input prompt to display</td>
<td>step</td>
</tr>
<tr>
<td>selection</td>
<td>display function selection msgs</td>
<td>off</td>
</tr>
<tr>
<td>set</td>
<td>show )set setting after assignment</td>
<td>off</td>
</tr>
<tr>
<td>startup</td>
<td>display messages on start-up</td>
<td>off</td>
</tr>
<tr>
<td>summary</td>
<td>print statistics after computation</td>
<td>off</td>
</tr>
<tr>
<td>testing</td>
<td>print system testing header</td>
<td>off</td>
</tr>
<tr>
<td>time</td>
<td>print timings after computation</td>
<td>off</td>
</tr>
<tr>
<td>type</td>
<td>print type after computation</td>
<td>on</td>
</tr>
<tr>
<td>void</td>
<td>print Void value when it occurs</td>
<td>off</td>
</tr>
<tr>
<td>any</td>
<td>print the internal type of objects of domain Any</td>
<td>on</td>
</tr>
<tr>
<td>naglink</td>
<td>show NAGLink messages</td>
<td>on</td>
</tr>
</tbody>
</table>

— messages —

```
(messages)
"show messages for various system features"
(interpreter)
(TREE)
(novar)
{
  \getchunk{messagesany}
  \getchunk{messagesautoload}
```
set message any

-------------- The any Option ---------------

Description: print the internal type of objects of domain Any

The any option may be followed by any one of the following:

-> on
   off

The current setting is indicated.

defvar $printAnyIfTrue

   — initvars —

(defvar |$printAnyIfTrue| t
   "print the internal type of objects of domain Any")
CHAPTER 25. SYSTEM COMMAND HANDLING

---

— messagesany —

(|any|
 "print the internal type of objects of domain Any"
 |interpreter|
 LITERALS
 |$printAnyIfTrue|
 (|on| |off|)
 |on|)

---

set message autoload

--------------------- The autoload Option ---------------------

Description: print file auto-load messages

defvar $printLoadMsgs

---

— initvars —

(defvar |$printLoadMsgs| nil "print file auto-load messages")

---

— messagesautoload —

(|autoload|
 "print file auto-load messages"
 |interpreter|
 LITERALS
 |$printLoadMsgs|
 (|on| |off|)
 |on|)

---
set message bottomup

--------------------- The bottomup Option ---------------------

Description: display bottom up modemap selection

The bottomup option may be followed by any one of the following:

  on
  -> off

The current setting is indicated.

defvar $reportBottomUpFlag

--- initvars ---

(defvar |$reportBottomUpFlag| nil "display bottom up modemap selection")

---

--- messagesbottomup ---

{|bottomup|
 "display bottom up modemap selection"
 |development| LITERALS
 |$reportBottomUpFlag| (|on| |off|)
 |off|)

---

set message coercion

--------------------- The coercion Option ---------------------

Description: display datatype coercion messages

The coercion option may be followed by any one of the following:
defvar $reportCoerceIfTrue

— initvars —

(defvar $reportCoerceIfTrue nil "display datatype coercion messages")

———

— messagescoercion —

(|coercion|
"display datatype coercion messages"
|development|
LITERALS
|$reportCoerceIfTrue|
(|on| |off|)
|off|)

———

set message dropmap

--------------------- The dropmap Option ---------------------

Description: display old map defn when replaced

The dropmap option may be followed by any one of the following:

on
-> off

The current setting is indicated.
defvar $displayDroppedMap

— initvars —

(defvar |$displayDroppedMap| nil "display old map defn when replaced")

---------

— messagesdropmap —

(dropmap|
 "display old map defn when replaced"
 (interpreter|
 LITERALS
 |$displayDroppedMap|
 (|on| |off|)
 |off|)

---------

set message expose

---------------------- The expose Option ----------------------

Description: warning for unexposed functions

The expose option may be followed by any one of the following:

  on
  -> off

The current setting is indicated.

defvar $giveExposureWarning

— initvars —

(defvar |$giveExposureWarning| nil "warning for unexposed functions")

---------
— messagesexpose —

(expose|
"warning for unexposed functions"
(interpreter|
LITERALS
($giveExposureWarning|
(|on| |off|)
|off|)

-----

set message file

----------------------- The file Option -----------------------

Description: print msgs also to SPADMSG LISTING

The file option may be followed by any one of the following:

   on
   -> off

The current setting is indicated.

defvar $printMsgsToFile

— initvars —

(defvar|$printMsgsToFile| nil "print msgs also to SPADMSG LISTING")

-----

— messagesfile —

(|file|
"print msgs also to SPADMSG LISTING"
(development|
LITERALS
($printMsgsToFile|
(|on| |off|)

)}
set message frame

--------------- The frame Option ---------------

Description: display messages about frames

The frame option may be followed by any one of the following:

on
-> off

The current setting is indicated.

defvar $frameMessages

—— initvars ——

(defvar $frameMessages nil "display messages about frames")

———

—— messagesframe ——

{|frame|
 "display messages about frames"
{|interpreter|
 LITERALS
 {|$frameMessages|
 (|on| |off|)
 |off|)

———

set message highlighting

--------------- The highlighting Option ---------------
Description: use highlighting in system messages

The highlighting option may be followed by any one of the following:

  on
  -> off

The current setting is indicated.

defvar $highlightAllowed

  — initvars —

(defvar |$highlightAllowed| nil "use highlighting in system messages")

———

  — messageshighlighting —

(|highlighting|
  "use highlighting in system messages"
  |interpreter|
  LITERALS
  |$highlightAllowed|
  (|on| |off|)
  |off|)

———

set message instant

--------------------- The instant Option ---------------------

Description: present instantiation summary

The instant option may be followed by any one of the following:

  on
  -> off

The current setting is indicated.
defvar $reportInstantiations

— initvars —

(defvar |$reportInstantiations| nil "present instantiation summary")

----------

— messagesinstant —

{|instant|
 "present instantiation summary"
|development|
LITERALS
|$reportInstantiations|
{|on| |off|}
|off|)

----------

set message insteach

--------------------- The insteach Option ---------------------

Description: present instantiation info

The insteach option may be followed by any one of the following:

  on
  -> off

The current setting is indicated.

defvar $reportEachInstantiation—

— initvars —

(defvar |$reportEachInstantiation| nil "present instantiation info")

----------
--- messagesinsideach ---

(`insideach` "present instantiation info" `development` LITERALS `$reportEachInstantiation` `off`) `off`)`insideach` "present instantiation info" `development` LITERALS `$reportEachInstantiation` `off`)

---

set message interponly

--------------- The interponly Option ---------------

Description: say when function code is interpreted

The interponly option may be followed by any one of the following:

-> on
   off

The current setting is indicated.

defvar $reportInterpOnly

--- initvars ---

(defvar `$reportInterpOnly` t "say when function code is interpreted")

defvar `$reportInterpOnly` t "say when function code is interpreted")

defvar `$reportInterpOnly` t "say when function code is interpreted")

--- messagesinterponly ---

(`interponly` "say when function code is interpreted" `interpreter` LITERALS `$reportInterpOnly` `off`) (`interponly` "say when function code is interpreted" `interpreter` LITERALS `$reportInterpOnly` `off`) (`interponly` "say when function code is interpreted" `interpreter` LITERALS `$reportInterpOnly` `off`)
set message naglink

---------------------- The naglink Option ----------------------

Description: show NAGLink messages

The naglink option may be followed by any one of the following:

-> on
  off

The current setting is indicated.

defvar $nagMessages

  — initvars —

(defvar |$nagMessages| t "show NAGLink messages")

  — messagesnaglink —

  (|naglink|
   "show NAGLink messages"
   |interpreter|
   LITERALS
   |$nagMessages|
   (|on| |off|)
   |on|)

  ———

set message number

---------------------- The number Option ----------------------
CHAPTER 25. SYSTEM COMMAND HANDLING

Description: display message number with message

The number option may be followed by any one of the following:

    on
    -> off

The current setting is indicated.

defvar $displayMsgNumber

    — initvars —

(defvar $displayMsgNumber nil "display message number with message")

    — messagesnumber —

  (number
   "display message number with message"
   (interpreter
    LITERALS
    ($displayMsgNumber
     (on |off))
    |off|)

set message prompt

---------------------- The prompt Option ----------------------

Description: set type of input prompt to display

The prompt option may be followed by any one of the following:

    none
    frame
    plain
    -> step
verbose

The current setting is indicated.

\texttt{defvar $inputPromptType}

\texttt{— initvars —}

\texttt{(defvar $inputPromptType 'step "set type of input prompt to display")}

\texttt{———}

\texttt{— messagesprompt —}

\texttt{|prompt|}
\texttt{$"set type of input prompt to display"}
\texttt{|interpreter|}
\texttt{LITERALS}
\texttt{$inputPromptType}
\texttt{(|none| |frame| |plain| |step| |verbose|)}
\texttt{|step|)

\texttt{———}

\texttt{set message selection}

\texttt{------------- The selection Option -------------------}

Description: display function selection msgs

The selection option may be followed by any one of the following:

\texttt{on}
\texttt{-} \texttt{off}

The current setting is indicated.

TPDHERE: This is a duplicate of \texttt{)set mes bot on} because both use the $reportBottomUpFlag flag

\texttt{—— messagesselection ——}
set

-------------- The set Option ---------------

Description: show )set setting after assignment

The set option may be followed by any one of the following:

  on
  -> off

The current setting is indicated.

defvar $displaySetValue

  --- initvars ---

(defvar $displaySetValue nil "show )set setting after assignment")

---

--- messageset ---

(|set|
  "show )set setting after assignment"
|interpreter|
LITERALS
|$displaySetValue|
(|on| |off|)
|off|)
set message startup

--------------------- The startup Option ---------------------

Description: display messages on start-up

The startup option may be followed by any one of the following:

  on
  --> off

The current setting is indicated.

defvar $displayStartMsgs

    — initvars —

(defvar |$displayStartMsgs| t "display messages on start-up")

    — messagesstartup —

  (|startup|
   "display messages on start-up"
   |interpreter|
   LITERALS
   |$displayStartMsgs|
   (|on| |off|)
   |on|)

set message summary

--------------------- The summary Option ---------------------

Description: print statistics after computation

The summary option may be followed by any one of the following:
CHAPTER 25. SYSTEM COMMAND HANDLING

on
-> off

The current setting is indicated.

defvar $printStatisticsSummaryIfTrue

— initvars —

(defvar $printStatisticsSummaryIfTrue| nil
  "print statistics after computation")

— messagesummary —

(|summary|
 "print statistics after computation"
 |interpreter|
 LITERALS
 |$printStatisticsSummaryIfTrue|
 (|on| |off|)
 |off|)

set message testing

--------------------- The testing Option ---------------------

Description: print system testing header

The testing option may be followed by any one of the following:

on
-> off

The current setting is indicated.
defvar $testingSystem

— initvars —

(defvar $testingSystem nil "print system testing header")

— message testing —

(|testing|
 "print system testing header"
 |development|
 LITERALS
 |on
 |off|)
 |off|)

------

set message time

----------------------- The time Option -----------------------

Description: print timings after computation

The time option may be followed by any one of the following:

on
-> off
  long

The current setting is indicated.

defvar $printTimeIfTrue

— initvars —

(defvar $printTimeIfTrue nil "print timings after computation")

------
--- messagestime ---

(defun messagestime
  (|time|
  "print timings after computation"
  |interpreter|
  LITERALS
  |$printTimeIfTrue|
  (|on| |off| |long|)
  |off|)

-------

set message type

------------------- The type Option -------------------

Description: print type after computation

The type option may be followed by any one of the following:

-> on
  off

The current setting is indicated.

defvar $printTypeIfTrue

--- initvars ---

(defun initvars
  (defvar |$printTypeIfTrue| t "print type after computation")

-------

--- messagetype ---

(defun messagetype
  (|type|
  "print type after computation"
  |interpreter|
  LITERALS
  |$printTypeIfTrue|
  (|on| |off|)
  |on|)

-------
set message void

------------------------------- The void Option -----------------------------

Description: print Void value when it occurs

The void option may be followed by any one of the following:

  on
-> off

The current setting is indicated.

defvar $printVoidIfTrue

--- initvars ---

(deffunc |$printVoidIfTrue| nil "print Void value when it occurs")

---

--- messagesvoid ---

(|void|
"print Void value when it occurs"
|interpreter|
LITERALS
|$printVoidIfTrue|
(|on| |off|)
|off|)

---

25.45 set naglink

Current Values of naglink Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>----------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
</tbody>
</table>
--- naglink ---

(|naglink|
 "options for NAGLink"
 |interpreter|
 TREE
 |novar|
 (\getchunk{naglinkhost}
 \getchunk{naglinkpersistence}
 \getchunk{naglinkmessages}
 \getchunk{naglinkdouble}
 ))

---------

set naglink host

----------------------- The host Option -----------------------

Description: internet address of host for NAGLink

)set naglink host is used to tell AXIOM which host to contact for a NAGLink request. An Internet address should be supplied. The host specified must be running the NAGLink daemon.

The current setting is localhost

defvar $nagHost

-- initvars --

(defvar |$nagHost| "localhost" "internet address of host for NAGLink")

---------

--- naglinkhost ---
25.45. **SET NAGLINK**

```
(defun setNagHost
  (declare (special $nagHost))
  (cond
    ((eq arg '|%initialize%|) (setq $nagHost "localhost"))
    ((eq arg '|%display%|) (object2String $nagHost))
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
      (describeSetNagHost))
    (t (setq $nagHost (object2String arg))))

(defun describeSetNagHost
  (sayBrightly (list "set naglink host "
    "is used to tell AXIOM which host to contact for")

(defun sayBrightly
  (list)
  "set naglink host "
```

(host)
"internet address of host for NAGLink "
(interpreter)
FUNCTION
(setNagHost)
(("enter host name"
  DIRECTORY
  $nagHost
  chkDirectory
  "localhost"))
NIL)
" a NAGLink request. An Internet address should be supplied. The host"
" specified must be running the NAGLink daemon."
" The current setting is 
|$nagHost|

---

set naglink persistence

------------------------- The persistence Option -------------------------

Description: number of (fortran) functions to remember

set naglink persistence is used to tell the nagd daemon how many ASP source and object files to keep around in case you reuse them. This helps to avoid needless recompilations. The number specified should be a non-negative integer.

The current setting is 1

defvar $fortPersistence

— initvars —

(defvar |$fortPersistence| 1 "number of (fortran) functions to remember")

——

— naglinkpersistence —

(|persistence|
 "number of (fortran) functions to remember 
|interpreter|
 FUNCTION
 |setFortPers|
 ("Requested remote storage (for asps):
 INTEGER
 |$fortPersistence|
 (0 NIL)
 10))
 NIL)
defun setFortPers

(describeFortPersistence p961)
sayMessage p961
[bright p961]
terminateSystemCommand p724
$fortPersistence p960

 — defun setFortPers —

(defun |setFortPers| (arg)
(let (n)
 (declare (special |$fortPersistence|))
 (cond
   ((eq arg '|%initialize%|) (setq |$fortPersistence| 1))
   ((eq arg '|%display%|) |$fortPersistence|)
   ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
     (describeFortPersistence))
   (t
    (setq n (car arg))
    (cond
     ((or (null (integerp n)) (minusp n))
      (sayMessage
       '("Your value of" ,@(|bright| n) "is invalid because ..."))
     (describeFortPersistence)
     (terminateSystemCommand))
    (t (setq |$fortPersistence| (car arg)))))))

———

defun describeFortPersistence

[sayBrightly p961]
$fortPersistence p960

 — defun describeFortPersistence —

(defun |describeFortPersistence| ()
 (declare (special |$fortPersistence|))
 (sayBrightly) (list
 "set naglink persistence "
 "is used to tell the "
 '|nagd|
 '| daemon how many ASP|
CHAPTER 25. SYSTEM COMMAND HANDLING

set naglink messages

----------------------------- The messages Option -----------------------------

Description: show NAGLink messages

The messages option may be followed by any one of the following:

* on
  * off

The current setting is indicated.

TPDHERE: this is the same as )set nag mes on

--- naglink messages ---

(|messages|
"show NAGLink messages"
|interpreter|
LITERALS
|$nagMessages|
(|on| |off|)
|on|)

set naglink double

------------------------ The double Option ------------------------

Description: enforce DOUBLE PRECISION ASPs
The double option may be followed by any one of the following:

-> on
  off

The current setting is indicated.

**defvar $nagEnforceDouble**

| initvars |

(defvar |$nagEnforceDouble| t "enforce DOUBLE PRECISION ASPs")

===

---

**naglinkdouble**

{double| "enforce DOUBLE PRECISION ASPs"
  |interpreter| LITERALS
  |$nagEnforceDouble|
  (|on| |off|)
  |on|)

---

### 25.46 set output

The result of the )set output command is:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>abbreviate</td>
<td>abbreviate type names</td>
<td>off</td>
</tr>
<tr>
<td>algebra</td>
<td>display output in algebraic form</td>
<td>On:CONSOLE</td>
</tr>
<tr>
<td>characters</td>
<td>choose special output character set</td>
<td>plain</td>
</tr>
<tr>
<td>fortran</td>
<td>create output in FORTRAN format</td>
<td>Off:CONSOLE</td>
</tr>
<tr>
<td>fraction</td>
<td>how fractions are formatted</td>
<td>vertical</td>
</tr>
<tr>
<td>html</td>
<td>create output in HTML style</td>
<td>Off:CONSOLE</td>
</tr>
<tr>
<td>length</td>
<td>line length of output displays</td>
<td>77</td>
</tr>
<tr>
<td>mathml</td>
<td>create output in MathML style</td>
<td>Off:CONSOLE</td>
</tr>
<tr>
<td>openmath</td>
<td>create output in OpenMath style</td>
<td>Off:CONSOLE</td>
</tr>
</tbody>
</table>
Since the output option has a bunch of sub-options each sub-option is defined within the output structure.

--- output ---

(set output
  "view and set some output options"
  [interpreter]
  TREE
  [novar]
  
  \getchunk{outputabbreviate}
  \getchunk{outputalgebra}
  \getchunk{outputcharacters}
  \getchunk{outputfortran}
  \getchunk{outputfraction}
  \getchunk{outputhtml}
  \getchunk{outputlength}
  \getchunk{outputmathml}
  \getchunk{outputopenmath}
  \getchunk{outputscript}
  \getchunk{outputscripts}
  \getchunk{outputshoweditor}
  \getchunk{outputtex}
  )
)

-----------

set output abbreviate

----------- The abbreviate Option -----------

Description: abbreviate type names

The abbreviate option may be followed by any one of the following:

  on
  -> off

The current setting is indicated.
defvar $abbreviateTypes

— initvars —

(deffunc |$abbreviateTypes| nil "abbreviate type names")

— outputabbreviate —

(defvar |$abbreviateTypes| nil "abbreviate type names")

(set output algebra
--------------------- The algebra Option ----------------------

Description: display output in algebraic form

)set output algebra is used to tell AXIOM to turn algebra-style output printing on and off, and where to place the output. By default, the destination for the output is the screen but printing is turned off.

Syntax: )set output algebra <arg>

where arg can be one of

|on| turn algebra printing on (default state)
|off| turn algebra printing off
|console| send algebra output to screen (default state)
|fp<.fe>| send algebra output to file with file prefix fp and file extension .fe. If not given, .fe defaults to .spout.

If you wish to send the output to a file, you may need to issue this command twice: once with on and once with the file name. For example, to send algebra output to the file polymer.spout, issue the two commands
(set output algebra on)
(set output algebra polymer)

The output is placed in the directory from which you invoked AXIOM or the one you set with the )cd system command.
The current setting is: On:CONSOLE

defvar $algebraFormat

— initvars —

(defvar |$algebraFormat| t "display output in algebraic form ")

———

defvar $algebraOutputFile

— initvars —

(defvar |$algebraOutputFile| "CONSOLE" "where algebra printing goes (enter \em console) or a pathname)?")

———

— outputalgebra —

(|algebra|
 "display output in algebraic form ">
|interpreter|
 FUNCTION
 |setOutputAlgebra|
 (|("display output in algebraic form 
 LITERALS
 |$algebraFormat| |
 |off| |on|)
 |on|)
 (break $algebraFormat)
 ("where algebra printing goes (enter \em console) or a pathname)?")
|FILENAME
 |$algebraOutputFile|
 |chkOutputFileName|
 "console")
NIL)
defvar $algebraOutputStream

--- initvars ---

(defun setOutputAlgebra

[defiostream p1128]
[concat p1197]
[describeSetOutputAlgebra p969]
[qcdr p??]
[qcar p??]
[member p1198]
[upcase p1206]
[sayKeyedMsg p27]
[shut p1128]
[pathnameType p1191]
[pathnameDirectory p1192]
[pathnameName p1191]
[$filep p??]
[make-outstream p1127]
<object2String p??]
[$algebraOutputStream p967]
[$algebraOutputFile p966]
[$filep p??]
[$algebraFormat p966]

--- defun setOutputAlgebra ---

(defun |setOutputAlgebra| (arg)
(let (label tmp1 tmp2 ptype fn ft fm filename teststream)
  (declare (special |$algebraOutputStream| |$algebraOutputFile| $filep $algebraFormat))
  (cond
    ((eq arg '|%initialize%|)
      (setq |$algebraOutputStream|
        (defiostream '((mode . output) (device . console)) 255 0))
      (setq |$algebraOutputFile| "CONSOLE")
      (setq $algebraFormat| t))
(eq arg '|%display%|
  (if $algebraFormat|
      (setq label "On:"
      (setq label "Off:"
      (concat label $algebraOutputFile|
        (or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
        (describeSetOutputAlgebra))
      t)
    (cond
      ((and (consp arg)
        (eq (qcdr arg) nil)
        (progn (setq fn (qcar arg)) t)
        (member fn '(y n ye yes no o on of off console
          |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
        '|ok|)
      (t (setq arg (list fn '|spout|)))))
    (cond
      ((and (consp arg)
        (eq (qcdr arg) nil)
        (progn (setq fn (qcar arg)) t))
      (cond
        ((member (upcase fn) '(y n ye o of))
          (sayKeyedMsg|
            (format nil
              "To toggle %1 printing on and off, specify %1 set output %2 ~
               yes/no/on/off %1 Yes, no, on and off cannot be abbreviated."
              '($algebra| $algebra|))
            (member (upcase fn) '(no off)) (setq $algebraFormat nil))
        ((member (upcase fn) '(yes on)) (setq $algebraFormat t))
        ((eq (upcase fn) 'console)
          (shut $algebraOutputStream|
            (setq $algebraOutputStream|
              (defiostream '((mode . output) (device . console)) 255 0))
              (setq $algebraOutputFile| "CONSOLE")))))
      (or
        (and (consp arg)
          (progn
            (setq fn (qcar arg))
            (setq tmp1 (qcdr arg))
            (and (consp tmp1)
              (eq (qcdr tmp1) nil)
              (progn (setq ft (qcar tmp1)) t))))
        (and (consp arg)
          (progn (setq fn (qcar arg))
            (setq tmp1 (qcdr arg))
            (and (consp tmp1)
              (progn (setq ft (qcar tmp1))
                (setq tmp2 (qcdr tmp1))
                (and (consp tmp2)
                  (eq (qcdr tmp2) nil)))
}
(progn
  (setq fm (qcar tmp2))
  t)))))

(when (setq ptype (|pathnameType| fn))
  (setq fn (concat (|pathnameDirectory| fn) (|pathnameName| fn)))
  (setq ft ptype))

(unless fm (setq fm 'a))

(setq filename ($filep fn ft fm))

(cond
  ((null filename)
   (|sayKeyedMsg| "It is not possible to open or create a file called %1 %2 %3 ."
                (list fn ft fm)))
  ((setq teststream (make-outstream filename 255 0))
   (shut |$algebraOutputStream|)
   (setq |$algebraOutputStream| teststream)
   (setq |$algebraOutputFile| (|object2String| filename))
   (|sayKeyedMsg| "%1 output will be written to file %2 ."
                (list "Algebra" |$algebraOutputFile|)))
  (t (|sayKeyedMsg| "It is not possible to open or create a file called %1 %2 %3 ."
                    (list fn ft fm)))))

(t
  (|sayKeyedMsg| "Your argument list is not valid." nil)
  (|describeSetOutputAlgebra|)))))))

defun describeSetOutputAlgebra

[sayBrightly p??]
[setOutputAlgebra p967]

— defun describeSetOutputAlgebra —

(defun |describeSetOutputAlgebra| ()
  (|sayBrightly| (list
    " set output algebra "
    "is used to tell AXIOM to turn algebra-style output"
    "printing on and off, and where to place the output. By default, the"
    "destination for the output is the screen but printing is turned off."
    "Syntax:  )set output algebra <arg>"
    " where arg can be one of"
    " on      turn algebra printing on (default state)"
    " off     turn algebra printing off")
  nil)
set output characters

------------------- The characters Option -------------------

Description: choose special output character set

The characters option may be followed by any one of the following:

  default
  -> plain

The current setting is indicated. This option determines the special characters used for algebraic output. This is what the current choice of special characters looks like:

  ulcer is shown as +        urc is shown as +
  lllc is shown as +        lrc is shown as +
  vbar is shown as |        hbar is shown as -
  quad is shown as ?        lbrk is shown as [   
  rbrk is shown as ]       lbrk is shown as {
  rbrkc is shown as }      ttee is shown as +
  btee is shown as +       rtee is shown as +
  ltee is shown as +       ctee is shown as +
  bslash is shown as \
--- outputcharacters ---

(defun setOutputCharacters
  (arg)
  (let (current char s l fn)
    (declare (special \$specialCharacters\ |$plainRTspecialCharacters| |$RTspecialCharacters| |$specialCharacterAlist|))
    (if (eq arg '|%initialize%|)
      (setq \$specialCharacters\ |$plainRTspecialCharacters| |$RTspecialCharacters| |$specialCharacterAlist|)
    (progn
      (setq current
        (cond
          ((eq \$specialCharacters\ |$RTspecialCharacters|) "default")
          ((eq \$specialCharacters\ |$plainRTspecialCharacters|) "plain")
          (t "unknown")))))
---

(defun setOutputCharacters
  (arg)
  (let (current char s l fn)
    (declare (special \$specialCharacters\ |$plainRTspecialCharacters| |$RTspecialCharacters| |$specialCharacterAlist|))
    (if (eq arg '|%initialize%|)
      (setq \$specialCharacters\ |$plainRTspecialCharacters| |$RTspecialCharacters| |$specialCharacterAlist|)
    (progn
      (setq current
        (cond
          ((eq \$specialCharacters\ |$RTspecialCharacters|) "default")
          ((eq \$specialCharacters\ |$plainRTspecialCharacters|) "plain")
          (t "unknown")))))
CHAPTER 25. SYSTEM COMMAND HANDLING

(set output fortran

--------------------- The fortran Option ---------------------

Description: create output in FORTRAN format

)set output fortran is used to tell AXIOM to turn FORTRAN-style output printing on and off, and where to place the output. By default, the destination for the output is the screen but printing is turned off.
Also See: \texttt{\&set fortran}

Syntax: \texttt{\&set output fortran <arg>}

where \texttt{arg} can be one of

\begin{itemize}
  \item \texttt{on} turn FORTRAN printing on
  \item \texttt{off} turn FORTRAN printing off (default state)
  \item \texttt{console} send FORTRAN output to screen (default state)
  \item \texttt{fp<.fe>} send FORTRAN output to file with file prefix
      \texttt{fp} and file extension \texttt{.fe}. If not given,
      \texttt{.fe} defaults to \texttt{.sfort}.
\end{itemize}

If you wish to send the output to a file, you must issue
this command twice: once with \texttt{on} and once with the file name.
For example, to send FORTRAN output to the file polymer.sfort,
issue the two commands

\begin{verbatim}
\&set output fortran on
\&set output fortran polymer
\end{verbatim}

The output is placed in the directory from which you invoked
AXIOM or the one you set with the \texttt{\&cd} system command.
The current setting is: \texttt{Off:CONSOLE}

\begin{verbatim}
defvar $fortranFormat
defvar $fortranOutputFile
\end{verbatim}
CHAPTER 25. SYSTEM COMMAND HANDLING

(defun setOutputFortran
  (arg)
  (let (label APPEND quiet tmp1 tmp2 ptype fn ft fm filename teststream)
(declare (special |$fortranOutputStream| |$fortranOutputFile| $filep
 |$fortranFormat|))

(cond
  ((eq arg '|%initialize%|)
   (setq |$fortranOutputStream|
         (defiostream '((mode . output) (device . console)) 255 0))
   (setq |$fortranOutputFile| "CONSOLE")
   (setq |$fortranFormat| nil))
  ((eq arg '|%display%|)
   (if |$fortranFormat|
       (setq label "On:"
             (setq label "Off:"))
       (concat label |$fortranOutputFile|))
   ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
    (|describeSetOutputFortran|))
  (t
   (DO ()
     (null (and (listp arg)
                 (member (upcase (car arg)) '(append quiet))))
     nil)
   (cond
    ((eq (upcase (car arg)) 'append) (setq append t))
    ((eq (upcase (car arg)) 'quiet) (setq quiet t))
    (t nil))
   (setq arg (cdr arg))))

(cond
  ((and (consp arg)
         (eq (qcdr arg) nil)
         (progn (setq fn (qcar arg)) t)
         (member fn '(Y N YE YES NO O ON OF OFF CONSOLE
                      |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
   'ok)
  (t (setq arg (list fn '|sfort|))))

(cond
  ((and (consp arg) (eq (qcdr arg) nil) (progn (setq fn (qcar arg)) t))
   (cond
    ((member (upcase fn) '(y n ye of))
     (sayKeyedMsg)
     (format nil
              "To toggle %1 printing on and off, specify %1 )set output %2 "
             (yes/no/on/off %1 Yes, no, on and off cannot be abbreviated.")
             (fortran |fortran|))
    ((member (upcase fn) '(no off)) (setq |$fortranFormat| nil))
    ((member (upcase fn) '(yes on)) (setq |$fortranFormat| t))
    ((eq (upcase fn) 'console)
     (shut |$fortranOutputStream|)
     (setq |$fortranOutputStream|
           (defiostream '((mode . output) (device . console)) 255 0))
     (setq |$fortranOutputFile| "CONSOLE")))
  (or
   
   (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
   (|describeSetOutputFortran|))

  ((null (and (listp arg)
              (member (upcase (car arg)) '(append quiet))))
   nil)
  (cond
   ((eq (upcase (car arg)) 'append) (setq append t))
   ((eq (upcase (car arg)) 'quiet) (setq quiet t))
   (t nil))
  (setq arg (cdr arg)))

(cond
  ((and (consp arg)
         (eq (qcdr arg) nil)
         (progn (setq fn (qcar arg)) t)
         (member fn '(Y N YE YES NO O ON OF OFF CONSOLE
                      |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
   'ok)
  (t (setq arg (list fn '|sfort|))))

(cond
  ((and (consp arg) (eq (qcdr arg) nil) (progn (setq fn (qcar arg)) t))
   (cond
    ((member (upcase fn) '(y n ye o of))
     (sayKeyedMsg)
     (format nil
              "To toggle %1 printing on and off, specify %1 )set output %2 "
             (yes/no/on/off %1 Yes, no, on and off cannot be abbreviated.")
             (fortran |fortran|))
    ((member (upcase fn) '(no off)) (setq |$fortranFormat| nil))
    ((member (upcase fn) '(yes on)) (setq |$fortranFormat| t))
    ((eq (upcase fn) 'console)
     (shut |$fortranOutputStream|)
     (setq |$fortranOutputStream|
           (defiostream '((mode . output) (device . console)) 255 0))
     (setq |$fortranOutputFile| "CONSOLE")))
  (or
   
   (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
   (|describeSetOutputFortran|))

  ((null (and (listp arg)
              (member (upcase (car arg)) '(append quiet))))
   nil)
  (cond
   ((eq (upcase (car arg)) 'append) (setq append t))
   ((eq (upcase (car arg)) 'quiet) (setq quiet t))
   (t nil))
  (setq arg (cdr arg)))

(cond
  ((and (consp arg)
         (eq (qcdr arg) nil)
         (progn (setq fn (qcar arg)) t)
         (member fn '(Y N YE YES NO O ON OF OFF CONSOLE
                      |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
   'ok)
  (t (setq arg (list fn '|sfort|))))

(cond
  ((and (consp arg) (eq (qcdr arg) nil) (progn (setq fn (qcar arg)) t))
   (cond
    ((member (upcase fn) '(y n ye o of))
     (sayKeyedMsg)
     (format nil
              "To toggle %1 printing on and off, specify %1 )set output %2 "
             (yes/no/on/off %1 Yes, no, on and off cannot be abbreviated.")
             (fortran |fortran|))
    ((member (upcase fn) '(no off)) (setq |$fortranFormat| nil))
    ((member (upcase fn) '(yes on)) (setq |$fortranFormat| t))
    ((eq (upcase fn) 'console)
     (shut |$fortranOutputStream|)
     (setq |$fortranOutputStream|
           (defiostream '((mode . output) (device . console)) 255 0))
     (setq |$fortranOutputFile| "CONSOLE")))
  (or
   
   (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
   (|describeSetOutputFortran|))

  ((null (and (listp arg)
              (member (upcase (car arg)) '(append quiet))))
   nil)
  (cond
   ((eq (upcase (car arg)) 'append) (setq append t))
   ((eq (upcase (car arg)) 'quiet) (setq quiet t))
   (t nil))
  (setq arg (cdr arg)))

(cond
  ((and (consp arg)
         (eq (qcdr arg) nil)
         (progn (setq fn (qcar arg)) t)
         (member fn '(Y N YE YES NO O ON OF OFF CONSOLE
                      |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
   'ok)
  (t (setq arg (list fn '|sfort|))))

(cond
  ((and (consp arg) (eq (qcdr arg) nil) (progn (setq fn (qcar arg)) t))
   (cond
    ((member (upcase fn) '(y n ye o of))
     (sayKeyedMsg)
     (format nil
              "To toggle %1 printing on and off, specify %1 )set output %2 "
             (yes/no/on/off %1 Yes, no, on and off cannot be abbreviated.")
             (fortran |fortran|))
    ((member (upcase fn) '(no off)) (setq |$fortranFormat| nil))
    ((member (upcase fn) '(yes on)) (setq |$fortranFormat| t))
    ((eq (upcase fn) 'console)
     (shut |$fortranOutputStream|)
     (setq |$fortranOutputStream|
           (defiostream '((mode . output) (device . console)) 255 0))
     (setq |$fortranOutputFile| "CONSOLE")))
  (or
   
   (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
   (|describeSetOutputFortran|))

  ((null (and (listp arg)
              (member (upcase (car arg)) '(append quiet))))
   nil)
  (cond
   ((eq (upcase (car arg)) 'append) (setq append t))
   ((eq (upcase (car arg)) 'quiet) (setq quiet t))
   (t nil))
  (setq arg (cdr arg)))

(cond
  ((and (consp arg)
         (eq (qcdr arg) nil)
         (progn (setq fn (qcar arg)) t)
         (member fn '(Y N YE YES NO O ON OF OFF CONSOLE
                      |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
   'ok)
  (t (setq arg (list fn '|sfort|))))
(and (consp arg)
  (progn
    (setq fn (qcar arg))
    (setq tmp1 (qcdr arg))
    (and (consp tmp1)
      (eq (qcdr tmp1) nil)
      (progn (setq ft (qcar tmp1)) t)))))

(and (consp arg)
  (progn
    (setq fn (qcar arg))
    (setq tmp1 (qcdr arg))
    (and (consp tmp1)
      (progn
        (setq ft (qcar tmp1))
        (setq tmp2 (qcdr tmp1))
        (and (consp tmp2)
          (eq (qcdr tmp2) nil)
          (progn (setq fm (qcar tmp2)) t))))))

(when (setq ptype (|pathnameType| fn))
  (setq fn (concat (|pathnameDirectory| fn) (|pathnameName| fn)))
  (setq ft ptype))

(unless fm (setq fm 'a))

(setq filename ($filep fn ft fm))
(cond
  ((null filename)
    (|sayKeyedMsg|
      "It is not possible to open or create a file called %1 %2 %3 ."
      (list fn ft fm))))
  ((setq teststream (|makeStream| append filename 255 0))
    (SHUT |$fortranOutputStream|)
    (setq |$fortranOutputStream| teststream)
    (setq |$fortranOutputFile| (|object2String| filename))
    (unless quiet
      (|sayKeyedMsg|
        "%1 output will be written to file %2 ."
        (list 'fortran |$fortranOutputFile|))))
  ((null quiet)
    (|sayKeyedMsg|
      "It is not possible to open or create a file called %1 %2 %3 ."
      (list fn ft fm)))
  (t nil))

(t
  (unless quiet (|sayKeyedMsg| "Your argument list is not valid." nil))
  (|describeSetOutputFortran|)))))))


set output fraction

--------------------- The fraction Option ---------------------

Description: how fractions are formatted
The fraction option may be followed by any one of the following:

- vertical
  horizontal

The current setting is indicated.

`defvar $fractionDisplayType`

--- initvars ---

`(defvar $fractionDisplayType 'vertical "how fractions are formatted")`  

--- outputfraction ---

`(|fraction|
  "how fractions are formatted"
  |interpreter|
  LITERALS
  |$fractionDisplayType|
  (|vertical| |horizontal|)
  |vertical|)`

---

`set output html`

----------------------- The html Option ------------------------

Description: create output in html style

)set output html is used to tell AXIOM to turn html-style output printing on and off, and where to place the output. By default, the destination for the output is the screen but printing is turned off.

Syntax:  )set output html <arg>
           where arg can be one of
           on        turn html printing on
           off       turn html printing off (default state)
           console   send html output to screen (default state)
set output html on
)set output html polymer

The output is placed in the directory from which you invoked Axiom or the one you set with the )cd system command.
The current setting is:  Off:CONSOLE

---

**defvar $htmlFormat**

---

```
(defvar |$htmlFormat| nil "create output in HTML format ")
```

---

**defvar $htmlOutputFile**

---

```
(defvar |$htmlOutputFile| "CONSOLE"
   "where HTML output goes (enter \em console) or a pathname")
```

---

---

```
(html)
 "create output in HTML style 
 |interpreter|
 FUNCTION
 |setOutputHtml|
 ("create output in HTML format 
 LITERALS
 |$htmlFormat|
```

---
defun setOutputHtml

(defvar $htmlOutputStream)  
(defvar $htmlOutputFile)  
(defvar $htmlFormat)  
(defvar $filep)

(defun setOutputHtml (arg)
  (let (label tmp1 tmp2 ptype fn ft fm filename teststream)
    (declare (special $htmlOutputStream $htmlOutputFile $htmlFormat $filep))
    (cond
      ((eq arg '|%initialize%|)
       (setq $htmlOutputStream
            (defiostream '((mode . output) (device . console)) 255 0))
       (setq $htmlOutputFile
             "CONSOLE")
       (setq $htmlFormat nil))
(setq label "On:"
(setq label "Off:"))
(concat label |$htmlOutputFile|))
(or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
(|describeSetOutputHtml|)
(t
(cond
((and (consp arg)
  (eq (qcdr arg) nil)
  (progn (setq fn (qcar arg)) t)
  (|member| fn '(y n ye yes no on off console
    |y| |n| |ye| |yes| |no| |on| |off| |console|)))
  '|ok|)
(t (setq arg (list fn '|smml|))))

(cond
((and (consp arg)
  (eq (qcdr arg) nil)
  (progn (setq fn (qcar arg)) t))
  (cond
    ((|member| (upcase fn) '(y n ye o of))
      (|sayKeyedMsg|
        format nil
        "To toggle %1 printing on and off, specify %1 set output %2 -
        yes/no/on/off %1 Yes, no, on and off cannot be abbreviated."
        '(|HTML| |html|)))
    ((|member| (upcase fn) '(no off)) (setq |$htmlFormat| nil))
    ((|member| (upcase fn) 'yes on)) (setq |$htmlFormat| t))
    (eq (upcase fn) 'console)
    (shut |$htmlOutputStream|)
    (setq |$htmlOutputStream|
      (defiostream '((mode . output) (device . console)) 255 0))
    (setq |$htmlOutputFile| "CONSOLE")))))
(or
(and (consp arg)
  (progn
    (setq fn (qcar arg))
    (setq tmp1 (qcdr arg))
    (and (consp tmp1)
      (eq (qcdr tmp1) nil)
      (progn (setq ft (qcar tmp1)) t))))
(and (consp arg)
  (progn
    (setq fn (qcar arg))
    (setq tmp1 (qcdr arg))
    (and (consp tmp1)
      (progn
        (setq ft (qcar tmp1))
        (setq tmp2 (qcdr tmp1))
        (and (consp tmp2)
(eq (qcdr tmp2) nil)
(progn
  (setq fm (qcar tmp2))
t)))))))

(when (setq ptype (|pathnameType| fn))
  (setq fn
    (concat (|pathnameDirectory| fn) (|pathnameName| fn)))
  (setq ft ptype))
(unless fm (setq fm 'a))
(setq filename ($filep fn ft fm))
(cond
  ((null filename)
    (sayKeyedMsg
      "It is not possible to open or create a file called %1 %2 %3 ."
      (list fn ft fm)))
  ((setq teststream (make-outstream filename 255 0))
    (shut |$htmlOutputStream|)
    (setq |$htmlOutputStream| teststream)
    (setq |$htmlOutputFile| (|object2String| filename))
    (sayKeyedMsg
      "%1 output will be written to file %2 ."
      (list "HTML" |$htmlOutputFile|)))
  (t (sayKeyedMsg
      "It is not possible to open or create a file called %1 %2 %3 ."
      (list fn ft fm)))))

(t
  (sayKeyedMsg!
    "Your argument list is not valid." nil)
  (|describeSetOutputHtml|))))))))

---

defun describeSetOutputHtml

[sayBrightly p??]
[setOutputHtml p980]

--- defun describeSetOutputHtml ---

(defun |describeSetOutputHtml| ()
  (|sayBrightly| (LIST
    "set output html"
    "is used to tell AXIOM to turn HTML-style output" 
    "printing on and off, and where to place the output. By default, the" 
    "destination for the output is the screen but printing is turned off."
    "Syntax: )set output html <arg>"
    "where arg can be one of"
set output length

---------------------- The length Option ----------------------

Description: line length of output displays

The length option may be followed by an integer in the range 10 to 245 inclusive. The current setting is 77

defvar $margin

— initvars —

(defvar $margin 3)

———

defvar $linelength

— initvars —
(defvar $linelength 77 "line length of output displays")

— outputlength —

(|length|
"line length of output displays"
|interpreter|
INTEGER
$LINELENGTH
(10 245)
77)

set output mathml

----------------------- The mathml Option ------------------------

Description: create output in MathML style

)set output mathml is used to tell AXIOM to turn MathML-style output
printing on and off, and where to place the output. By default,
the destination for the output is the screen but printing is
turned off.

Syntax: )set output mathml <arg>
    where arg can be one of
    on    turn MathML printing on
    off   turn MathML printing off (default state)
    console send MathML output to screen (default state)
    fp<.fe> send MathML output to file with file prefix fp
            and file extension .fe. If not given,
            .fe defaults to .smml.

If you wish to send the output to a file, you must issue
this command twice: once with on and once with the file name.
For example, to send MathML output to the file polymer.smml,
issue the two commands

)set output mathml on
)set output mathml polymer

The output is placed in the directory from which you invoked
AXIOM or the one you set with the )cd system command.
The current setting is: Off:CONSOLE

defvar $mathmlFormat

    — initvars —

(defvar $mathmlFormat nil "create output in MathML format ")

defvar $mathmlOutputFile

    — initvars —

(defvar $mathmlOutputFile "CONSOLE"
"where MathML output goes (enter \em console\) or a pathname")

    — outputmathml —

(|mathml|
"create output in MathML style "
|interpreter|
FUNCTION
|setOutputMathml|
("create output in MathML format 
LITERALS
|$mathmlFormat|
(|off| |on|)
|off|
(|break||$mathmlFormat|)
("where MathML output goes (enter \em console\) or a pathname")
FILENAME
|$mathmlOutputFile|
|chkOutputFileName|
"console")
NIL)

    — —
defun setOutputMathml

(defun setOutputMathml (arg)
  (let (label tmp1 tmp2 ptype fn ft fm filename teststream)
    (declare (special $mathmlOutputStream $mathmlOutputFile $mathmlFormat $filep))
    (cond
      ((eq arg '|%initialize%|)
       (setq $mathmlOutputStream
         (defiostream '((mode . output) (device . console)) 255 0))
       (setq $mathmlOutputFile "CONSOLE")
       (setq $mathmlFormat nil))
      ((eq arg '|%display%|)
       (if $mathmlFormat
         (setq label "On:"
           (concat label $mathmlOutputFile))
         (setq label "Off:"
           (concat label $mathmlOutputFile))))
      (t
        (concat label "%")
        (describeSetOutputMathml))))
    (t
      (cond
        ((and (consp arg)
          (eq (qcdr arg) nil)
          (progn (setq fn (qcar arg)) t)
          (member fn '(y n ye yes no o on of off console)
            |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|))))
25.46. **SET OUTPUT**

```
'(ok)
(t (setq arg (list fn 'sml)))))
(cond
  ((and (consp arg)
         (eq (qcdr arg) nil)
         (progn (setq fn (qcar arg)) t))
    (cond
     ((member (upcase fn) '(y n ye o of))
      (sayKeyedMsg)
      (format nil
        "To toggle %1 printing on and off, specify %1 set output %2 ~
        yes/no/on/off %1 Yes, no, on and off cannot be abbreviated."
        fn arg arg)
      (member (upcase fn) '(no off)) (setq $mathmlFormat nil))
     ((member (upcase fn) '(yes on)) (setq $mathmlFormat t))
     ((eq (upcase fn) 'console)
      (shut $mathmlOutputStream)
      (setq $mathmlOutputStream
        (defiostream '((mode . output) (device . console)) 255 0))
      (setq $mathmlOutputFile "CONSOLE")))
    (or
     ((and (consp arg)
           (setq fn (qcar arg))
           (setq tmp1 (qcdr arg))
           (and (consp tmp1)
                (eq (qcdr tmp1) nil)
                (progn (setq ft (qcar tmp1)) t))))
     (setq ptype (pathnameType fn))
     (setq fn
       (concat (pathnameDirectory fn) (pathnameName fn)))
     (unless fm (setq fm 'a))
     (setq filename ($filep fn ft fm))
     (cond
      (null filename)
      (sayKeyedMsg)
      "It is not possible to open or create a file called %1 %2 %3 ."
    )))
```


(list fn ft fm)))
((setq teststream (make-outstream filename 255 0))
 (shut |$mathmlOutputStream|)
 (setq |$mathmlOutputStream| teststream)
 (setq |$mathmlOutputFile| (|object2String| filename))
 (|sayKeyedMsg| "%1 output will be written to file %2 ."
 (list "MathML" |$mathmlOutputFile|)))
 (t (|sayKeyedMsg| "It is not possible to open or create a file called %1 %2 %3 ."
 (list fn ft fm))))
(t
 (|sayKeyedMsg| "Your argument list is not valid." nil)
 (|describeSetOutputMathml|)))))))

defun describeSetOutputMathml

[sayBrightly p??]
[setOutputMathml p986]
— defun describeSetOutputMathml —

(defun |describeSetOutputMathml| ()
 (|sayBrightly| (LIST
 " )set output mathml "
 "is used to tell AXIOM to turn MathML-style output"
 '|%l| "printing on and off, and where to place the output. By default, the"
 '|%l| "destination for the output is the screen but printing is turned off."
 '|%l|
 '|%l| "Syntax: )set output mathml <arg>"
 '|%l| " where arg can be one of"
 '|%l| " on turn MathML printing on"
 '|%l| " off turn MathML printing off (default state)"
 '|%l| " console send MathML output to screen (default state)"
 '|%l| " fp<.fe> send MathML output to file with file prefix fp and file"
 '|%l| " extension .fe. If not given, .fe defaults to .stex."
 '|%l|
 '|%l| "If you wish to send the output to a file, you must issue this command"
 '|%l| "twice: once with"
 " on and once with the file name. For example, to send"
 '|%l| "MathML output to the file polymer.smml, issue the two commands"
 '|%l|
 '|%l| " )set output mathml on"
 '|%l| " )set output mathml polymer"
 '|%l|
set output openmath

----------------------- The openmath Option ------------------------

Description: create output in OpenMath style

)set output tex is used to tell AXIOM to turn OpenMath output printing on and off, and where to place the output. By default, the destination for the output is the screen but printing is turned off.

Syntax: )set output tex <arg>
where arg can be one of
  on          turn OpenMath printing on
  off         turn OpenMath printing off (default state)
  console     send OpenMath output to screen (default state)
  fp<.fe>     send OpenMath output to file with file prefix fp and file extension .fe. If not given, .fe defaults to .sopen.

If you wish to send the output to a file, you must issue this command twice: once with on and once with the file name.
For example, to send OpenMath output to the file polymer.sopen, issue the two commands

)set output openmath on
)set output openmath polymer

The output is placed in the directory from which you invoked AXIOM or the one you set with the )cd system command.
The current setting is: Off:CONSOLE

defvar $openMathFormat

— initvars —

(defvar |$openMathFormat| nil "create output in OpenMath format ")
defvar $openMathOutputFile

— initvars —

(defvar $openMathOutputFile "CONSOLE"
 "where TeX output goes (enter {em console} or a pathname)"
)

— outputOpenmath —

(openmath
 "create output in OpenMath style "
 interpreter
 FUNCTION
 setOutputOpenMath
 (("create output in OpenMath format "
 LITERALS
 ($openMathFormat
 (|off| |on|)
 |off|)
 (|break| $openMathFormat|)
 ("where TeX output goes (enter {em console} or a pathname)"
 FILENAME
 ($openMathOutputFile
 |chkOutputFileName|)
 "console")
 NIL)

defun setOutputOpenMath

[defistream p1128]
[concat p1197]
[describeSetOutputOpenMath p993]
[qcdr p779]
[qcar p779]
[member p1198]
[upcase p1206]
[sayKeyedMsg p27]
setoutputopenmath

(defun |setOutputOpenMath| (arg)
  (let ((label tmp1 tmp2 ptype fn ft fm filename teststream)
    (declare (special |$openMathOutputStream| |$openMathFormat| $filep |
      |$openMathOutputFile|)))
    (cond
      ((eq arg '|%initialize%|)
        (setq |$openMathOutputStream|
          (defiostream '((mode . output) (device . console)) 255 0))
        (setq |$openMathOutputFile| "CONSOLE")
        (setq |$openMathFormat| NIL))
      ((eq arg '|%display%|)
        (if |$openMathFormat|
          (setq label "On:"
          (setq label "Off:"))
          (concat label |$openMathOutputFile|)))
      ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
        (!describeSetOutputOpenMath|))
      (t
        (cond
          ((and (consp arg)
              (eq (qcdr arg) nil)
              (progn (setq fn (qcar arg)) t)
              (|member| fn 'y n ye yes no o on of off console
                |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
            'ok)
          (t (setq arg (list fn '|som|))))
        (cond
          ((and (consp arg)
              (eq (qcdr arg) nil)
              (progn (setq fn (qcar arg)) t))
            (cond
              ((|member| (upcase fn) '(y n ye o of))
                (!sayKeyedMsg|)
                (format nil
                  "To toggle %1 printing on and off, specify %1 set output %2 ~")
          |ok|)
          (t (setq arg (list fn '|som|))))
        )
      )
    )
  )

— defun setOutputOpenMath —
yes/no/on/off %1 Yes, no, on and off cannot be abbreviated.

'(OpenMath |openmath|)
(((member (upcase fn) '(no off)) (setq $openMathFormat nil))
(((member (upcase fn) '(yes on)) (setq $openMathFormat t))
((eq (upcase fn) 'console)
 (shut $openMathOutputStream)
 (setq $openMathOutputStream
 (defiostream '((mode . output) (device . console)) 255 0))
 (setq $openMathOutputFile "CONSOLE"))))
((or
  (and (consp arg)
   (progn (setq fn (qcar arg))
     (setq tmp1 (qcdr arg))
     (and (consp tmp1)
       (eq (qcdr tmp1) nil)
       (progn (setq ft (qcar tmp1)) t))))
 (and (consp arg)
   (progn
    (setq fn (qcar arg))
    (setq tmp1 (qcdr arg))
    (and (consp tmp1)
     (progn (setq ft (qcar tmp1))
       (setq tmp2 (qcdr tmp1))
       (and (consp tmp2)
        (setq fm (qcar tmp2))
        (setq fm 'a))
     (progn (setq fm (qcar tmp2)) t))))
    (when (setq ptype (pathnameType fn))
     (setq fn (concat (pathnameDirectory fn) (pathnameName fn))
     (setq ft ptype))
     (unless fm (setq fm 'a))
     (setq filename ($filep fn ft fm))
     (cond
      ((null filename)
       (sayKeyedMsg
        "It is not possible to open or create a file called %1 %2 %3 ."
        (list fn ft fm))
      (setq teststream (make-outstream filename 255 0))
      (shut $openMathOutputStream)
      (setq $openMathOutputStream teststream)
      (setq $openMathOutputFile (object2String filename))
      (sayKeyedMsg
       "%1 output will be written to file %2 ."
       (list "OpenMath" $openMathOutputFile)))
      (t
       (sayKeyedMsg
        "It is not possible to open or create a file called %1 %2 %3 ."
        (list fn ft fm)))))
      (t
       (sayKeyedMsg ("Your argument list is not valid." nil)
       (describeSetOutputOpenMath))))))

992  

CHAPTER 25. SYSTEM COMMAND HANDLING  

yes/no/on/off %1 Yes, no, on and off cannot be abbreviated.

'(OpenMath |openmath|)
(((member (upcase fn) '(no off)) (setq $openMathFormat nil))
(((member (upcase fn) '(yes on)) (setq $openMathFormat t))
((eq (upcase fn) 'console)
 (shut $openMathOutputStream)
 (setq $openMathOutputStream
 (defiostream '((mode . output) (device . console)) 255 0))
 (setq $openMathOutputFile "CONSOLE"))))
((or
  (and (consp arg)
   (progn (setq fn (qcar arg))
     (setq tmp1 (qcdr arg))
     (and (consp tmp1)
       (eq (qcdr tmp1) nil)
       (progn (setq ft (qcar tmp1)) t))))
 (and (consp arg)
   (progn
    (setq fn (qcar arg))
    (setq tmp1 (qcdr arg))
    (and (consp tmp1)
     (progn (setq ft (qcar tmp1))
       (setq tmp2 (qcdr tmp1))
       (and (consp tmp2)
        (setq fm (qcar tmp2))
        (setq fm 'a))
     (progn (setq fm (qcar tmp2)) t))))
    (when (setq ptype (pathnameType fn))
     (setq fn (concat (pathnameDirectory fn) (pathnameName fn))
     (setq ft ptype))
     (unless fm (setq fm 'a))
     (setq filename ($filep fn ft fm))
     (cond
      ((null filename)
       (sayKeyedMsg
        "It is not possible to open or create a file called %1 %2 %3 ."
        (list fn ft fm))
      (setq teststream (make-outstream filename 255 0))
      (shut $openMathOutputStream)
      (setq $openMathOutputStream teststream)
      (setq $openMathOutputFile (object2String filename))
      (sayKeyedMsg
       "%1 output will be written to file %2 ."
       (list "OpenMath" $openMathOutputFile)))
      (t
       (sayKeyedMsg
        "It is not possible to open or create a file called %1 %2 %3 ."
        (list fn ft fm)))))
      (t
       (sayKeyedMsg ("Your argument list is not valid." nil)
       (describeSetOutputOpenMath))))))
defun describeSetOutputOpenMath

[sayBrightly p??]
[setOutputOpenMath p990]

— defun describeSetOutputOpenMath —

(defun |describeSetOutputOpenMath| ()
  (|sayBrightly| (list
    "set output openmath"
    "is used to tell AXIOM to turn OpenMath output"
    "printing on and off, and where to place the output. By default, the"
    "destination for the output is the screen but printing is turned off."
    "Syntax: )set output openmath <arg>"
    "where arg can be one of"
    "on turn OpenMath printing on"
    "off turn OpenMath printing off (default state)"
    "console send OpenMath output to screen (default state)"
    "fp<.fe> send OpenMath output to file with file prefix fp and file"
    "extension .fe. If not given, .fe defaults to .som."
    "If you wish to send the output to a file, you must issue this command"
    "twice: once with"
    "on and once with the file name. For example, to send"
    "OpenMath output to the file polymer.som, issue the two commands"
    ")set output openmath on"
    ")set output openmath polymer"
    "The output is placed in the directory from which you invoked AXIOM or"
    "the one you set with the )cd system command."
    "The current setting is:"
    (|setOutputOpenMath| '(%display%))
  )))

set output script

---------------------- The script Option ----------------------
Description: display output in SCRIPT formula format

)set output script is used to tell AXIOM to turn IBM Script formula-style output printing on and off, and where to place the output. By default, the destination for the output is the screen but printing is turned off.

Syntax:  )set output script <arg>
         where arg can be one of
         on  turn IBM Script formula printing on
         off turn IBM Script formula printing off
             (default state)
         console send IBM Script formula output to screen
             (default state)
         fp<.fe> send IBM Script formula output to file with file prefix fp and file extension .fe. If not given, .fe defaults to .sform.

If you wish to send the output to a file, you must issue this command twice: once with on and once with the file name. For example, to send IBM Script formula output to the file polymer.sform, issue the two commands

   )set output script on
   )set output script polymer

The output is placed in the directory from which you invoked AXIOM or the one you set with the )cd system command.

The current setting is: Off:CONSOLE

defvar $formulaFormat

   — initvars —

(defvar |$formulaFormat| nil "display output in SCRIPT format")

defvar $formulaOutputFile

   — initvars —

(defvar |$formulaOutputFile| "CONSOLE")
where script output goes (enter \emph{console} or a a pathname)"

| outputscript |

\begin{verbatim}
(defun setOutputFormula
  [deostream p1128]
  [concat p1197]
  [describeSetOutputFormula p997]
  [qcdr p??]
  [qcar p??]
  [member p1198]
  [upcase p1206]
  [sayKeyedMsg p27]
  [shut p1128]
  [pathnameType p1191]
  [pathnameDirectory p1192]
  [pathnameName p1191]
  [$filep p??]
  [make-outstream p1127]
  [object2String p??]
  [$formulaOutputStream p??]
  [$formulaOutputFile p994]
\end{verbatim}
(defun setOutputFormula (arg)
  (let ((label tmp1 tmp2 ptype fn ft fm filename teststream)
        (declare (special $formulaOutputStream $formulaOutputFile $filep
                        $formulaFormat)))
    (cond
      ((eq arg '|%initialize%|)
       (setq $formulaOutputStream
             (defiostream '((mode . output) (device . console)) 255 0))
       (setq $formulaOutputFile "CONSOLE")
       (setq $formulaFormat nil))
      ((eq arg '|%display%|)
       (if $formulaFormat
           (setq label "On:")
           (setq label "Off:"))
       (concat label $formulaOutputFile))
      ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
           (describeSetOutputFormula))
      (t
       (cond
        ((and (consp arg)
               (eq (qcdr arg) nil))
         (progn
          (setq fn (qcar arg))
          (ok)
          (t (setq arg (list fn '|sform|))))
        (cond
         ((member (upcase fn) '(y n ye yes o of))
          (sayKeyedMsg)
          (format nil
                   "To toggle %1 printing on and off, specify %l set output %2  
                    yes/no/on/off %l Yes, no, on and off cannot be abbreviated."))
         ((member (upcase fn) '(no off)) (setq $formulaFormat nil))
         ((member (upcase fn) '(yes on)) (setq $formulaFormat t))
         ((eq (upcase fn) 'console)
          (SHUT $formulaOutputStream)
          (setq $formulaOutputStream
                (defiostream '((mode . output) (device . console)) 255 0))
          (setq $formulaOutputFile "CONSOLE")))
         (t (or
             )))
    )))
(and (consp arg)
  (progn (setq fn (qcar arg))
    (setq tmp1 (qcdr arg))
    (and (consp tmp1)
      (eq (qcdr tmp1) nil)
      (progn (setq ft (qcar tmp1)) t))))

(and (consp arg)
  (progn (setq fn (qcar arg))
    (setq tmp1 (qcdr arg))
    (and (consp tmp1)
      (progn (setq ft (qcar tmp1))
        (setq tmp2 (qcdr tmp1))
        (and (consp tmp2)
          (eq (qcdr tmp2) nil)
          (progn
            (setq fm (qcar tmp2)) t)))))

(if (setq ptype (|pathnameType| fn))
  (setq fn (concat (|pathnameDirectory| fn) (|pathnameName| fn)))
  (setq ft ptype))

(unless fm (setq fm 'a))

(setq filename ($filep fn ft fm))

(cond
  ((null filename)
   (|sayKeyedMsg| "It is not possible to open or create a file called %1 %2 %3 ."
    (list fn ft fm))
  ((setq teststream (make-outstream filename 255 0))
   (shut |$formulaOutputStream|)
   (setq |$formulaOutputStream| teststream)
   (setq |$formulaOutputFile| (|object2String| filename))
   (|sayKeyedMsg| "%1 output will be written to file %2 ."
     (list "IBM Script formula" |$formulaOutputFile| )))
  (t
   (|sayKeyedMsg| "It is not possible to open or create a file called %1 %2 %3 ."
     (list fn ft fm)))))

(defun describeSetOutputFormula
  [sayBrightly p??]
  [setOutputFormula p995]
CHAPTER 25. SYSTEM COMMAND HANDLING

— defun describeSetOutputFormula —

(defun |describeSetOutputFormula| ()
  (|sayBrightly| (list
   " )set output script
   "is used to tell AXIOM to turn IBM Script formula-style"
   "output printing on and off, and where to place the output. By default, the" 
   "destination for the output is the screen but printing is turned off."
   "Syntax: )set output script <arg>
   " where arg can be one of"
   " on turn IBM Script formula printing on"
   " off turn IBM Script formula printing off (default state)"
   " console send IBM Script formula output to screen (default state)"
   " fp<.fe> send IBM Script formula output to file with file prefix fp"
   " and file extension .fe. If not given, .fe defaults to .sform."
   "If you wish to send the output to a file, you must issue this command"
   "twice: once with"
   " on and once with the file name. For example, to send"
   "IBM Script formula output to the file polymer.sform,"
   "issue the two commands"
   " )set output script on"
   " )set output script polymer"
   " The output is placed in the directory from which you invoked AXIOM or"
   "the one you set with the )cd system command."
   "The current setting is: "
  )|setDisplay|
))

set output scripts

---------------------- The scripts Option ----------------------

Description: show subscripts,... linearly

The scripts option may be followed by any one of the following:

yes
no
The current setting is indicated.

\texttt{defvar \$linearFormatScripts}

\begin{verbatim}
| initvars |

(defvar |\$linearFormatScripts| nil "show subscripts,... linearly")
\end{verbatim}

\begin{verbatim}
-----
| outputscripts |

{|scripts| "show subscripts,... linearly" |interpreter| LITERALS |\$linearFormatScripts| (|on| |off|) |off|)
\end{verbatim}

\begin{verbatim}
-----
set output showeditor
\end{verbatim}

\begin{verbatim}
--------------------- The showeditor Option ---------------------

Description: view output of )show in editor

The showeditor option may be followed by any one of the following:

on -> off

The current setting is indicated.
\end{verbatim}

\texttt{defvar \$useEditorForShowOutput}

\begin{verbatim}
| initvars |
\end{verbatim}
(defvar $useEditorForShowOutput nil "view output of )show in editor")

---

outputshoweditor

(|showeditor|
 "view output of )show in editor"
 |interpreter|
 LITERALS
 |$useEditorForShowOutput|
 (|on| |off|)
 |off|)

---

set output tex

----------------------- The tex Option ------------------------

Description: create output in TeX style

)set output tex is used to tell AXIOM to turn TeX-style output printing on and off, and where to place the output. By default, the destination for the output is the screen but printing is turned off.

Syntax: )set output tex <arg>

where arg can be one of

on turn TeX printing on
off turn TeX printing off (default state)
console send TeX output to screen (default state)
fp<.fe> send TeX output to file with file prefix fp and file extension .fe. If not given, .fe defaults to .stex.

If you wish to send the output to a file, you must issue this command twice: once with on and once with the file name. For example, to send TeX output to the file polymer.stex, issue the two commands

)set output tex on
)set output tex polymer

The output is placed in the directory from which you invoked AXIOM or the one you set with the )cd system command.

The current setting is: Off:CONSOLE
defvar $texFormat

    — initvars —

(defvar $texFormat nil "create output in TeX format ")

----------

defvar $texOutputFile

    — initvars —

(defvar $texOutputFile "CONSOLE"
    "where TeX output goes (enter \em console) or a pathname")

----------

    — outputtex —

({\text
    "create output in TeX style 
    \text{interpreter}
    FUNCTION
    \text{setOutputTex}
    (("create output in TeX format 
        LITERALS
        $\text{texFormat}$
        (|off| |on|)
        |off|
        (|break| $\text{texFormat}$)
        ("where TeX output goes (enter \em console) or a pathname")
        FILENAME
        $\text{texOutputFile}$
        |chkOutputFileName| "console")
    NIL)

----------

defun setOutputTex

[defiostream p1128]
[concat p1197]
(defun setOutputTex (arg)
  (let (label tmp1 tmp2 ptype fn ft fm filename teststream)
    (declare (special \$texOutputStream \$texOutputFile \$texFormat $filep))
    (cond
      ((eq arg '|%initialize%|)
       (setq \$texOutputStream
         (defiostream '((mode . output) (device . console)) 255 0))
       (setq \$texOutputFile| "CONSOLE")
       (setq \$texFormat| nil))
      ((eq arg '|%display%|)
       (if \$texFormat| (setq label "On:")
         (setq label "Off:"))
       (concat label \$texOutputFile|))
      ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
       (|describeSetOutputTex|))
      (t
       (cond
        ((and (consp arg)
               (eq (qcdr arg) nil)
               (progn (setq fn (qcar arg)) t)
               (|member| fn '(|y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
         '|ok|)
        (t (setq arg (list fn '|stex| nil))))
    (cond
      ((and (consp arg)
            (eq (qcdr arg) nil)))
      (t nil))
  )
  )
(progn (setq fn (qcar arg)) t))
(cond
  (\(\text{|member| \ (upcase fn) \ '(y n ye o of)}\))
    (\text{|sayKeyedMsg|}
      (format nil
        "To toggle %1 printing on and off, specify %1 )set output %2 ~
        yes/no/on/off %1 Yes, no, on and off cannot be abbreviated."))
    '((\text{TeX} |tex|)))
  (\(\text{|member| \ (upcase fn) \ '(no off)}\) (setq \$\text{texFormat| nil}))
  (\(\text{|member| \ (upcase fn) \ '(yes on)}\) (setq \$\text{texFormat| t}))
  (\(\text{eq (upcase fn) \ 'console}\)
    (shut \$\text{texOutputStream|})
    (setq \$\text{texOutputStream|}
      (defiostream '((mode . output) (device . console)) 255 0))
    (setq \$\text{texOutputFile| "CONSOLE")))
  (or
    (and (consp arg)
      (progn (setq fn (qcar arg))
        (setq tmp1 (qcdr arg))
        (and (consp tmp1)
          (eq (qcdr tmp1) nil)
          (progn (setq ft (qcar tmp1)) t)))))
  (and (consp arg)
    (progn (setq fn (qcar arg))
      (setq tmp1 (qcdr arg))
      (and (consp tmp1)
        (eq (qcdr tmp1) nil)
        (progn (setq ft (qcar tmp1)) t)))))
  (when (setq ptype (\text{|pathnameType| fn}))
    (setq fn (concat (\text{|pathnameDirectory| fn}) (\text{|pathnameName| fn}))
      (setq ft ptype))
  (unless fm (setq fm 'A))
  (setq filename (\text{|filep fn ft fm|})
  (cond
    ((null filename)
      (\text{|sayKeyedMsg|}
        "It is not possible to open or create a file called %1 %2 %3 .")
      (list fn ft fm)))
    ((setq teststream (\text{make-outstream filename 255 0}))
      (shut \$\text{texOutputStream|})
      (setq \$\text{texOutputStream| teststream)
      (setq \$\text{texOutputFile| (\text{|object2String| filename}))
      (\text{|sayKeyedMsg|}
        "%1 output will be written to file %2 .")
      (list "\text{TeX}" \$\text{texOutputFile|})
      (t (\text{|sayKeyedMsg|}
        "It is not possible to open or create a file called %1 %2 %3 .")
(list fn ft fm )))
(t
  (|sayKeyedMsg| "Your argument list is not valid." nil)
  (|describeSetOutputTex|))))))))

——

defun describeSetOutputTex

[|sayBrightly| p??]
[|setOutputTex| p1001]

—— defun describeSetOutputTex ——

(defun |describeSetOutputTex| ()
  (|sayBrightly| (list
    " )set output tex 
  "is used to tell AXIOM to turn TeX-style output"
    '|%l| "printing on and off, and where to place the output. By default, the"
    '|%l| "destination for the output is the screen but printing is turned off."
    '|%l|
    '|%l| "Syntax: )set output tex <arg>"
    '|%l| " where arg can be one of"
    '|%l| " on turn TeX printing on"
    '|%l| " off turn TeX printing off (default state)"
    '|%l| " console send TeX output to screen (default state)"
    '|%l| " fp<.fe> send TeX output to file with file prefix fp and file"
    '|%l| " extension .fe. If not given, .fe defaults to .stex."
    '|%l|
    '|%l| "If you wish to send the output to a file, you must issue this command"
    '|%l| " twice: once with"
    " on and once with the file name. For example, to send"
    '|%l| "TeX output to the file polymer.stex, issue the two commands"
    '|%l|
    '|%l| " )set output tex on"
    '|%l| " )set output tex polymer"
    '|%l|
    '|%l| "The output is placed in the directory from which you invoked AXIOM or"
    '|%l| " the one you set with the )cd system command."
    '|%l| " The current setting is: 
  (|setOutputTex| '|%display%|) )
))))

——
25.47 quit

----------------------- The quit Option -----------------------

Description: protected or unprotected quit

The quit option may be followed by any one of the following:

   protected
-> unprotected

The current setting is indicated.

defvar $quitCommandType

   — initvars —

(defvar |$quitCommandType| 'unprotected| "protected or unprotected quit")

---

   — quit —

(|quit|
 "protected or unprotected quit"
 |interpreter|
 LITERALS
 |$quitCommandType|
 (|protected| |unprotected|)
 |unprotected|)

---

25.48 streams

Current Values of streams Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>calculate</td>
<td>specify number of elements to calculate</td>
<td>10</td>
</tr>
<tr>
<td>showall</td>
<td>display all stream elements computed</td>
<td>off</td>
</tr>
</tbody>
</table>
set streams calculate

--------------------- The calculate Option ---------------------

Description: specify number of elements to calculate

)set streams calculate is used to tell AXIOM how many elements of a stream to calculate when a computation uses the stream. The value given after calculate must either be the word all or a positive integer.

The current setting is 10.

defvar $streamCount

— initvars —

(defvar |$streamCount| 10
   "number of initial stream elements you want calculated")

— streamscalculate —

(|calculate|
 "specify number of elements to calculate ")
|interpreter|
FUNCTION
|setStreamsCalculate|
defun setStreamsCalculate

(let (n)
  (declare (special $streamCount))
  (cond
    ((eq arg '|%initialize%|) (setq $streamCount 10))
    ((eq arg '|%display%|) (object2String $streamCount))
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
      (describeSetStreamsCalculate))
    (t
      (setq n (car arg))
      (cond
        ((and (not (eq n '|all|)) (or (null (integerp n)) (minusp n)))
         (sayMessage
          '"Your value of" ,0([bright] n) "is invalid because ..."'))
        (describeSetStreamsCalculate)
        (terminateSystemCommand)))
    (t (setq $streamCount n))))))

— defun setStreamsCalculate —

defun describeSetStreamsCalculate

(sayKeyedMsg $streamCount)
set streams showall

------------- The showall Option -------------

Description: display all stream elements computed

The showall option may be followed by any one of the following:

  on
  -> off

The current setting is indicated.

defvar $streamsShowAll

--- initvars ---

(defvar |$streamsShowAll| nil "display all stream elements computed")

---

--- streamsshowall ---

(|showall|
  "display all stream elements computed"
  |interpreter|
  LITERALS
  |$streamsShowAll|)
25.49 set system

Current Values of system Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>functioncode</td>
<td>show gen. LISP for functions when compiled</td>
<td>off</td>
</tr>
<tr>
<td>optimization</td>
<td>show optimized LISP code</td>
<td>off</td>
</tr>
<tr>
<td>prettyprint</td>
<td>prettyprint BOOT func's as they compile</td>
<td>off</td>
</tr>
</tbody>
</table>

---

(set some system development variables)

---

set system functioncode

------------- The functioncode Option ---------------

Description: show gen. LISP for functions when compiled

The functioncode option may be followed by any one of the following:

   on
         -> off

The current setting is indicated.
defvar $reportCompilation

— initvars —

(defvar |$reportCompilation| nil "show gen. LISP for functions when compiled")

——

— systemfunctioncode —

(|functioncode|
 "show gen. LISP for functions when compiled"
 |development|
 LITERALS
 |$reportCompilation|
 (|on| |off|)
 |off|)

——

set system optimization

----------------------- The optimization Option -----------------------

Description: show optimized LISP code

The optimization option may be followed by any one of the following:

    on
    -> off

The current setting is indicated.

defvar $reportOptimization

— initvars —

(defvar |$reportOptimization| nil "show optimized LISP code")

——
— systemoptimization —

(optimization
 "show optimized LISP code"
 (development
 LITERALS
 ($reportOptimization|
 (on|off|)
 (off|)
)

———

set system prettyprint

------------------- The prettyprint Option -------------------

Description: prettyprint BOOT func’s as they compile

The prettyprint option may be followed by any one of the following:

on
  -> off

The current setting is indicated.

defvar $prettyprint

— initvars —

(defvar $prettyprint t "prettyprint BOOT func’s as they compile")

———

— systemprettyprint —

($prettyprint|
 "prettyprint BOOT func’s as they compile"
 (development
 LITERALS
 $prettyprint
 (on|off|)
)
25.50 set userlevel

------------------- The userlevel Option -------------------

Description: operation access level of system user

The userlevel option may be followed by any one of the following:

    interpreter
    compiler
    -> development

The current setting is indicated.

defvar $UserLevel
    — initvars —
    (defvar $UserLevel "development" "operation access level of system user")

    — userlevel —
    ($UserLevel
     "operation access level of system user"
     interpreter
     LITERALS
     $UserLevel
     (interpreter compiler development)
     development))

    — initvars —
(defvar |$setOptions| '(
\getchunk{breakmode}
\getchunk{compile}
\getchunk{debug}
\getchunk{expose}
\getchunk{functions}
\getchunk{fortran}
\getchunk{kernel}
\getchunk{hyperdoc}
\getchunk{help}
\getchunk{history}
\getchunk{messages}
\getchunk{naglink}
\getchunk{output}
\getchunk{quit}
\getchunk{streams}
\getchunk{system}
\getchunk{userlevel}
))}

--------

defvar $setOptionNames

— initvars —

(defvar |$setOptionNames| (mapcar #'car |$setOptions|))

--------

— postvars —

(eval-when (eval load)
  (|initializeSetVariables| |$setOptions|))

--------

25.51 Set code

defun set

[set1 p1014]
[$setOptions p??]
### defun set

```lisp
(defun set (l)
  (declare (special $setOptions$))
  (set1 l $setOptions$))
```

---

**defun set1**

This function will be called with the top level arguments to )set. For instance, given the command

```
)set break break
```

this function gets

```
(set1 (|break| |break|) ....)
```

and given the command

```
)set mes auto off
```

this function gets

```
(set1 (|mes| |auto| |off|) ....)
```

which, because “message” is a TREE, generates the recursive call:

```
(set1 (|auto| |off|) <the message subtree>)
```

The “autoload” subtree contains a FUNCTION called printLoadMessages, which gets called with %describe%

*displaySetVariableSettings p897*

*seq p??*

*exit p??*

*selectOption p751*

*downcase p1206*

*lassoc p??*

*satisfiesUserLevel p723*

*sayKeyedMsg p27*

*poundsign p??*

*displaySetOptionInformation p896*

*sayMSG p29*
(defun set1 (l settree)
  (let ((|$setOptionNames| arg setdata st setfunarg num upperlimit arg2)
        (declare (special |$setOptionNames| |$UserLevel| |$displaySetValue|)))
    (cond
     ((null l) (|displaySetVariableSettings| settree '||))
     (t
      (setq |$setOptionNames|
            (do ((t1 settree (cdr t1)) t0 (x nil))
                 ((or (atom t1) (progn (setq x (car t1)) nil)) (nreverse0 t0))
               (seq
                (exit
                 (setq t0 (cons (elt x 0) t0)))))))
    (setq arg
          (|selectOption| (downcase (car l)) |$setOptionNames| '|optionError|))
    (setq setdata (cons arg (lassoc arg settree)))
    (cond
     ((null (|satisfiesUserLevel| (third setdata)))
      (|sayKeyedMsg| "Your %1 is ambiguous. The following are abbreviated by %2 :
                      (list |$UserLevel| "set option" nil)))
     ((eql 1 (|#| l)) (|displaySetOptionInformation| arg setdata))
     (t
      (setq st (fourth setdata))
      (case (fourth setdata)
        (function
         (setq setfunarg
               (if (eq (elt l 1) 'default)
                    '|%initialize%|
                    (ifcdr l)))
          (if (canFuncall? (fifth setdata))
              (funcall (fifth setdata) setfunarg)
              (|sayMSG| (concatenate 'string " Function not implemented. "
                                 (string (fifth setdata)))))))
     (when |$displaySetValue|
          (|displaySetOptionInformation| arg setdata)))
  )
NIL)
(string
(setq arg2 (elt 1 1))
(cond
  ((eq arg2 'default) (set (fifth setdata) (seventh setdata)))
  (arg2 (set (fifth setdata) arg2))
  (t nil))
(when (or \$displaySetValue\ (null arg2))
  (\displaySetOptionInformation\ arg setdata))
NIL)
(integer
(setq arg2
  (progn
    (setq num (elt 1 1))
    (cond
      ((and (integerp num)
          (>= num (elt (sixth setdata) 0)))
       (or (null (setq upperlimit (elt (sixth setdata) 1)))
           (<= num upperlimit)))
       num)
      (t
       (\selectOption\ (elt 1 1)
        (cons \default\ (sixth setdata)) nil)))))
(cond
  ((eq arg2 'default) (set (fifth setdata) (seventh setdata)))
  (arg2 (set (fifth setdata) arg2))
  (t nil))
(when (or \$displaySetValue\ (null arg2))
  (\displaySetOptionInformation\ arg setdata))
(when (null arg2)
  (\sayMessage\ "Your value", (\object2String\ (elt 1 1)))
  "is not among the valid choices.")
  (t nil)))
(literals
(cond
  ((setq arg2
        (\selectOption\ (elt 1 1)
         (cons \default\ (sixth setdata)) nil))
   (cond
     ((eq arg2 'default)
      (set (fifth setdata)
           (\translateYesNo2TrueFalse\ (seventh setdata))))
     (t
      (cond ((eq arg2 \nobreak\)
             #+:GCL (\use-fast-links t)))
      (cond...
((eq arg2'|fastlinks|)
  #+GCL (si::use-fast-links nil)
  (setq arg2'|break|)))
  (set (fifth setdata) (translateYesNo2TrueFalse arg2)))))))
  (when (or |$displaySetValue| (null arg2))
    (|displaySetOptionInformation| arg setdata))
  (cond
    ((null arg2)
      (|sayMessage|
        (cons " Your value"
          (append ([bright] ([object2String] (elt l 1)))
          (cons "is not among the valid choices." nil)))))
      (t nil)))
  (tree ([set1| (ifcdr l) (sixth setdata)] nil)
  (t
    (|sayMessage|
      "Cannot handle set tree node type" ,@([bright] [yet])
      nil)))))))

        
25.52  )show Command

    show man page

--- show.help ---

=================================
A.22. )show
=================================

User Level Required: interpreter

Command Syntax:

- )show nameOrAbbrev
- )show nameOrAbbrev )operations
- )show nameOrAbbrev )attributes

Command Description:
This command displays information about AXIOM domain, package and category
constructors. If no options are given, the )operations option is assumed. For
example,

)show POLY
)show POLY )operations
)show Polynomial
)show Polynomial )operations

each display basic information about the Polynomial domain constructor and
then provide a listing of operations. Since Polynomial requires a Ring (for
example, Integer) as argument, the above commands all refer to a unspecified
ring R. In the list of operations, $ means Polynomial(R).

The basic information displayed includes the signature of the constructor
(the name and arguments), the constructor abbreviation, the exposure status
of the constructor, and the name of the library source file for the
constructor.

If operation information about a specific domain is wanted, the full or
abbreviated domain name may be used. For example,

)show POLY INT
)show POLY INT )operations
)show Polynomial Integer
)show Polynomial Integer )operations

are among the combinations that will display the operations exported by the
domain Polynomial(Integer) (as opposed to the general domain constructor
Polynomial). Attributes may be listed by using the \texttt{attributes} option.

\begin{verbatim}
Also See:
  o \texttt{display}
  o \texttt{set}
  o \texttt{what}
\end{verbatim}

\begin{verbatim}
defun The \texttt{show} command

\texttt{[showSpad2Cmd p1019]}

\texttt{— defun show —}

(defun \texttt{show} (arg) (\texttt{[showSpad2Cmd arg]})

\end{verbatim}

\begin{verbatim}
defun The internal \texttt{show} command

\texttt{[member p1198]}
\texttt{[helpSpad2Cmd p813]}
\texttt{[sayKeyedMsg p27]}
\texttt{[qcar p??]}
\texttt{[reportOperations p1021]}
\texttt{[$showOptions p??]}
\texttt{[$e p247]}
\texttt{[$env p247]}
\texttt{[$InteractiveFrame p23]}
\texttt{[$options p??]}

\texttt{— defun showSpad2Cmd —}

(defun \texttt{[showSpad2Cmd arg]}
(let (\texttt{[$showOptions |$e |$env | constr]})
  (declare (special \texttt{[$showOptions |$e |$env |$InteractiveFrame |$options]})
  (if (equal arg (list nil))
    (\texttt{[helpSpad2Cmd \texttt{'(\texttt{show})}]})
    (progn
      (setq \texttt{[$showOptions \texttt{'(\texttt{attributes |operations])}]})
\end{verbatim}

\footnote{“display” (25.18 p 796) “set” (25.51 p 1013) “what” (25.62 p 1091)}
(unless $options| (setq $options| '((|operations|)))))
(setq $e| $InteractiveFrame|)
(setq $env| $InteractiveFrame|)
(cond
  ((and (consp arg) (eq (qcdr arg) nil) (progn (setq constr (qcar arg)) t))
   (cond
    ((|member| constr '([|Union| |Record| |Mapping|]))
      (cond
        ((eq constr '|Record|)
          (|sayKeyedMsg|)
          (format nil
            "Record(a:A,...,b:B) %l Record takes any number of ~
            selector-domain pairs as arguments: %i %l a, a selector, an ~
            element of domain Symbol %l A, a domain of category ~
            SetCategory %l ... %l b, a selector, an element of domain ~
            Symbol %l B, a domain of category SetCategory %u %l ~
            This constructor is a primitive in Axiom. ~
            The selectors a,...,b of a Record type must be distinct. %l %l ~
            In order for more information to be displayed about %1 , ~
            you must give it specific arguments. For example: %2 %l ~
            You can also use the HyperDoc Browser.")
        (list constr ")show Record(a: Integer, b: String)" ))
      ))
    ((eq constr '|Map|eling|)
      (|sayKeyedMsg|)
      (format nil
        "Mapping(T, S, ...) %l Mapping takes any number of arguments ~
        of the form: %i %l T, a domain of category SetCategory %l ~
        S, a domain of category SetCategory %l ... %u %l ~
        Mapping(T, S, ...) denotes the class of objects which are ~
        mappings from a source domain (S, ...) into a target domain T. ~
        The Mapping constructor can take any number of arguments. ~
        All but the first argument is regarded as part of a source ~
        tuple for the mapping. For example, Mapping(T, A, B) denotes ~
        the class of mappings from (A, B) into T. %l ~
        This constructor is a primitive in Axiom. ~
        For more information, use the HyperDoc Browser.")
        (list constr "")
      )))
    (t
      (|sayKeyedMsg|)
      (format nil
        "Tagged union: Union(a:A, ..., b:B) %l Union takes any number ~
        of 'tag'-domain pairs of arguments: %i %l a, a tag, an ~
        element of domain Symbol %l A, a domain of category ~
        SetCategory %l ... %l b, a tag, an element of domain ~
        Symbol %l B, a domain of category SetCategory %u %l ~
        This constructor is a primitive in Axiom. ~
        In this tagged Union, tags a, ..., b must be distinct. %l %l ~
        In order for more information to be displayed about %1 , ~
        you must give it specific arguments. For example: %2 %l ~
        You can also use the HyperDoc Browser.")
      ))
    )
  ))
)
Untagged union: Union(A, ..., B) %l Union takes any number ~  
of domain arguments: %l %l A, a domain of category ~  
SetCategory %l ... %l B, a domain of category SetCategory %u %l ~  
In this untagged form of Union, domains A, ..., B must be ~  
distinct. In order for more information to be displayed about ~  
%1, you must give it specific arguments. For example: %2 %l ~  
You can also use the HyperDoc Browser.

Mapping(T, S, ...) %l Mapping takes any number of arguments ~  
of the form: %l %l T, a domain of category SetCategory %l ~  
S, a domain of category SetCategory %l ... %u %l ~  
Mapping(T, S, ...) denotes the class of objects which are ~  
mappings from a source domain (S, ...) into a target domain T. ~  
The Mapping constructor can take any number of arguments. ~  
All but the first argument is regarded as part of a source ~  
tuple for the mapping. For example, Mapping(T, A, B) denotes ~  
the class of mappings from (A, B) into T. %l ~  
This constructor is a primitive in Axiom. ~  
For more information, use the HyperDoc Browser.

---

defun reportOperations

[sayBrightly p?]
[bright p?]
[sayKeyedMsg p27]
[qcar p?]
isNameOfType p?]
isDomainValuedVariable p1105]
[reportOpsFromUnitDirectly0 p1027]
[opOf p?]
[unabbrev p?]
[reportOpsFromLisplib0 p1023]
evaluateType p1078]
[mkAtree p?]
[removeZeroOneDestructively p?]
isType p?]
--- defun reportOperations ---

(defun reportOperations (oldArg u)
  (let ([$env] [$$eval] [$$genValue] [$$doNotAddEmptyModeIfTrue]
         tmp1 v unitForm tree unitFormp)
    (declare (special $$env $$eval $$genValue $$quadSymbol
                     $$doNotAddEmptyModeIfTrue))
    (setq $$env (list (list nil)))
    (setq $$eval t)
    (setq $$genValue t)
    (when u
      (setq $$doNotAddEmptyModeIfTrue t)
      (cond
       ((equal u $$quadSymbol)
        (sayBrightly (cons " mode denotes" (append ([bright] "any") (list '|type|))))
       ((eq u '%)
        (sayKeyedMsg
         (format nil "The )show system command is used to display information about ~
                     types or partial types. For example, )show Integer will show ~
                     information about Integer."
         nil)
       (sayKeyedMsg
        (format nil "%l %% is a special variable holding the result of the last ~
                     computation. Issue )display properties %% to see this value.")))
      ((and (null (and (consp u) (eq (qcar u) '|Record|)))
        (null (and (consp u) (eq (qcar u) '|Union|)))
        (null ([isNameOfType] u))
        (null (and (consp u)
                   (eq (qcar u) '|typeOf|))
            (progn
             (setq tmp1 (qcdr u))
             (and (consp tmp1) (eq (qcdr tmp1) nil))))))
      (when (atom oldArg) (setq oldArg (list oldArg))
       (sayKeyedMsg
        (format nil "The )show system command is used to display information about ~
                     types or partial types. For example, )show Integer will show ~
                     information about Integer.")))
    nil)
(dolist (op oldArg)  
  (sayKeyedMsg)  
  (format nil  
    "%s %s is not the name of a known type constructor. If you want ~  
    to see information about any operations named %s, issue ~  
    %ceon )display operations %s %ceoff")  
  (list (opOf op))))  
((setq v (isDomainValuedVariable u)) (reportOpsFromUnitDirectly0 v))  
(t  
  (if (atom u)  
    (setq unitForm (opOf (unabbrev u)))  
    (setq unitForm (unabbrev u)))  
  (if (atom unitForm)  
    (reportOpsFromLisplib0 unitForm u)  
    (progn  
      (setq unitFormP (evaluateType unitForm))  
      (setq tree (mkAtree (removeZeroOneDestructively unitForm)))  
      (if (setq unitFormP (isType tree))  
        (reportOpsFromUnitDirectly0 unitFormP)  
        (sayKeyedMsg)  
          (format nil  
            "It is not known what %s is, so no information about it can be ~  
            displayed.")  
          (list unitForm))))))))

defun reportOpsFromLisplib0

[reportOpsFromLisplib1 p1024]  
[reportOpsFromLisplib p1024]  
[$useEditorForShowOutput p999]

— defun reportOpsFromLisplib0 —

(defun |reportOpsFromLisplib0| (unitForm u)  
  (declare (special |$useEditorForShowOutput|))  
  (if |$useEditorForShowOutput|  
    (|reportOpsFromLisplib1| unitForm u)  
    (|reportOpsFromLisplib| unitForm u)))

— —
defun reportOpsFromLisplib1

(defun reportOpsFromLisplib1 (unitForm u)
  (let (|$sayBrightlyStream| showFile)
    (declare (special |$sayBrightlyStream| $erase))
    (setq showFile (|pathname| (list 'show 'listing 'a)))
    ($erase showFile)
    (setq |$sayBrightlyStream|
      (defiostream `((file ,showFile) (mode . output)) 255 0))
    (|sayShowWarning|)
    (reportOpsFromLisplib u)
    (shut |$sayBrightlyStream|)
    (|editFile| showFile)))

—— defun reportOpsFromLisplib1 ——

defun reportOpsFromLisplib

(defun reportOpsFromLisplib (constructor? p??)
  (sayKeyedMsg p27)
  (getConstructorSignature p??)
  (ifcdr p??)
  (getdatabase p1156)
  (eqsubstlist p??)
  (nreverse0 p??)
  (sayBrightly p??)
  (concat p1197)
  (bright p??)
  (form2StringWithWhere p??)
  (isExposedConstructor p1026)
  (strconc p??)
  (namestring p1190)
  (selectOptionLC p751)
defun reportOpsFromLisplib (op u)
(let (fn s nArgs argList functorForm argml tmp1 functorFormWithDecl
  verb sourceFile opt attList)
  (declare (special $linelength $showOptions $options
    $FormalMapVariableList))
  (if (null (setq fn (constructor? op)))
    (sayKeyedMsg "%1 is unknown, so no information is available." (list u))
    (progn
      (setq argml (when (setq s (getConstructorSignature op)) (ifcdr s)))
      (setq typ (getdatabase op 'constructorkind))
      (setq nArgs (cdr argml))
      (setq argList (ifcdr (getdatabase op 'constructorform))
        (setq functorForm (cons op argList))
        (setq argumentList (eqsubstlist argList $FormalMapVariableList argml))
        (mapcar #'(lambda (a m) (push (list '|:| a m) tmp1)) argList argml)
        (setq functorFormWithDecl (cons op (nreverse0 tmp1)))
        (sayBrightly)
        (concat (bright (form2StringWithWhere functorFormWithDecl))
          " is a" (bright typ) "constructor")
        (sayBrightly)
        (cons " Abbreviation for" (append (bright op) (cons "is" (bright fn)))))))
  (if (isExposedConstructor op)
    (setq verb "is")
    (setq verb "is not"))
  (sayBrightly)
  (cons " This constructor" (append (bright verb) (list "exposed in this frame.")))
  (setq sourceFile (getdatabase op 'sourcefile))
  (sayBrightly)
  (cons " Issue" (append (bright (strconc "edit " (namestring sourceFile)))
    (cons "to see algebra source code for" (append (bright fn) (list '|%l|))))))
(dolist (item |$options|)
  (setq opt (|selectOptionLC| (car item) |$showOptions| '|optionError|))
  (cond
    ((eq opt '|layout|) (|dc1| fn))
    ((eq opt '|views|)
      (|sayBrightly|
        (cons "To get" (append (|bright| "views")
          (list "you must give parameters of constructor"))))
    ((eq opt '|attributes|)
      (format t "-v,,-:Q<"a"->:"%" (- $linelength 2) " Attributes ")
      (|sayBrightly| "")
      (setq attList
        (remdup
          (msort
            (mapcar #'(lambda (x) (caar x))
              (reverse (getdatabase op 'attributes))))))
      (if (null attList)
        (|sayBrightly| (concat (|form2String| functorForm) '|has no attributes.| '|%|))
        (|say2PerLine| (mapcar #'|formatAttribute| attList))))
    ((eq opt '|operations|)
      (displayOperationsFromLisplib functorForm))))))

defun isExposedConstructor

[getalist p??]
|$localExposureData p98|
|$globalExposureGroupAlist p99|

— defun isExposedConstructor —

(defun |isExposedConstructor| (name)
  (let (x found)
    (declare (special |$globalExposureGroupAlist| |$localExposureData|))
    (cond
      ((member name '(|Union| |Record| |Mapping|)) t)
      ((member name (elt |$localExposureData| 2)) nil)
      ((member name (elt |$localExposureData| 1)) t)
      (t
        (loop for g in (elt |$globalExposureGroupAlist| 0)
          when (not found)
          do
            (setq x (getalist |$globalExposureGroupAlist| g))
            (when (and x (getalist x name)) (setq found t))))
defun displayOperationsFromLisplib

(let (name argl kind opList opl ops)
  (setq name (car form))
  (setq argl (cdr form))
  (setq kind (getdatabase name 'constructorkind))
  (format t "~v,,,'-:@<~a~>~%" (- $linelength 2) " Operations ")
  (setq opList (getdatabase name 'operationalist))
  (if (null opList)
    (reportOpsFromUnitDirectly form)
    (progn
      (setq opl (remdup (msort (eqsubstlist argl $FormalMapVariableList opList))))
      (setq ops nil)
      (dolist (x opl)
        (setq ops (append ops (formatOperationAlistEntry x))))
      (say2PerLine ops))))

defun reportOpsFromUnitDirectly0

(progn
  (reportOpsFromUnitDirectly1 form)
  (reportOpsFromUnitDirectly form))
— defun reportOpsFromUnitDirectly0 —

(defun |reportOpsFromUnitDirectly0| (D)
  (declare (special |$useEditorForShowOutput|))
  (if |$useEditorForShowOutput|
      (|reportOpsFromUnitDirectly1| D)
      (|reportOpsFromUnitDirectly| D)))

———

defun reportOpsFromUnitDirectly

[member p1198]
[qcar p??]
[evalDomain p1075]
[getdatabase p1156]
[sayBrightly p??]
[concat p1197]
[formatOpType p??]
[isExposedConstructor p1026]
[bright p??]
[sayBrightly p??]
[strconc p??]
[namestring p1190]
[selectOptionLC p751]
[specialChar p1126]
[remdup p??]
[msort p??]
[formatAttribute p??]
[getl p1200]
[systemErrorHere p??]
[nreverse0 p??]
[getOplistForConstructorForm p1030]
[say2PerLine p??]
[formatOperation p??]
[$commentedOps p??]
[$CategoryFrame p??]
[$lineLength p983]
[$options p??]
[$showOptions p??]

— defun reportOpsFromUnitDirectly —

(defun |reportOpsFromUnitDirectly| (unitForm)
(let ((\$commentedOps\ isRecordOrUnion unit top kind abb sourceFile verb opt
  attList constructorFunction tmp1 funlist a sigList tmp2)
  (declare (special \$commentedOps\ \$CategoryFrame\ $linelength \$options\ \$showOptions\))
  (setq isRecordOrUnion
    (and (consp unitForm)
      (progn (setq a (qcar unitForm)) t)
        (member a '(Record Union)))))
  (setq unit (evalDomain unitForm))
  (setq top (car unitForm))
  (setq kind (getdatabase top 'constructorkind))
  (sayBrightly)
    (concat " " (formatOpType unitForm)
      " is a " kind " constructor.")
  (unless isRecordOrUnion
    (setq abb (getdatabase top 'abbreviation))
    (setq sourceFile (getdatabase top 'sourcefile))
    (sayBrightly)
      (cons " Abbreviation for"
        (append (bright top) (cons "is" (bright abb)))))
    (if (isExposedConstructor top)
      (setq verb "is")
        (setq verb "is not"))
    (sayBrightly)
      (cons " This constructor"
        (append (bright verb) (list "exposed in this frame." ))))
    (sayBrightly)
      (cons " Issue"
        (append (bright (strconc )edit " (namestring sourceFile)))
          (cons "to see algebra source code for"
            (append (bright abb) (list '|%) g))))
  (dolist (item \$options\)
    (setq opt (selectOptionLC (car item) \$showOptions\ 'optionError))
      (cond
        ((eq opt '|attributes|)
          (format t "v,,,'-:<a>-%" (- $linelength 2) " Attributes ")
        (if isRecordOrUnion
          (sayBrightly) " Records and Unions have no attributes.")
        (progn
          (sayBrightly)
            (setq attList
              (remdup
                (msort
                  (mapcar #'(lambda (unit2) (car unit2)) (reverse (elt unit 2))))))
                (say2PerLine)
                  (mapcar #'(formatAttribute attList)
                    nil))
          ((eq opt '|operations|)
            (setq \$commentedOps\ 0)
              ; --new form is (<op> <signature> <slotNumber> <condition> <kind>)

defun getOplistForConstructorForm

The new form is an op-Alist which has entries

  (<op> . signature-Alist)

where signature-Alist has entries

  (<signature> . item)

where item has form (slotNumber; condition; kind)

  (<slotNumber> <condition> <kind>)

where kind = ELT | CONST | Subsumed | (XLAM..)

  <kind> = ELT | CONST | Subsumed | (XLAM..)
— defun getOplistForConstructorForm —

(defun getOplistForConstructorForm (form)
  (let (argl pairlis opAlist op signatureAlist result)
    (declare (special $FormalMapVariableList))
    (setq op (car form))
    (setq argl (cdr form))
    (setq pairlis
      (loop for fv in $FormalMapVariableList
        for arg in argl
          collect (cons fv arg)))
    (setq opAlist (getOperationAlistFromLisplib op))
    (loop for item in opAlist do
      (setq op (car item))
      (setq signatureAlist (cdr item))
      (setq result
        (append result
          (getOplistWithUniqueSignatures op pairlis signatureAlist))))
    result)

——

defun getOplistWithUniqueSignatures

— defun getOplistWithUniqueSignatures —

(defun getOplistWithUniqueSignatures (op pairlis signatureAlist)
  (let (sig slotNumber pred kind alist)
    (loop for item in signatureAlist
      when (not (eq (fourth item) 'Subsumed))
      do
        (setq sig (first item))
        (setq slotNumber (second item))
        (setq pred (third item))
        (setq kind (fourth item))
        (setq alist
          (insertAlist
            (sublis pairlis (list op sig))
            (sublis pairlis (list pred (list kind nil slotNumber)))
            alist))
    alist)

——
defun reportOpsFromUnitDirectly1

[pathname p1192]
[erase p??]
[defiostream p1128]
[sayShowWarning p1032]
[reportOpsFromUnitDirectly p1028]
[shut p1128]
[editFile p805]
[$sayBrightlyStream p??]
[$erase p??]

— defun reportOpsFromUnitDirectly1 —

(defun reportOpsFromUnitDirectly1 (D)
  (let (($sayBrightlyStream| showFile)
    (declare (special $sayBrightlyStream| $erase))
    (setq showFile (pathname (list 'show 'listing 'a)))
    ($erase showFile)
    (setq !$sayBrightlyStream|
      (defiostream '((file ,showFile) (mode . output)) 255 0))
    ($sayShowWarning|)
    ($reportOpsFromUnitDirectly| D)
    (shut !$sayBrightlyStream|)
    ($editFile| showFile)))

—

defun sayShowWarning

[sayBrightly p??]

— defun sayShowWarning —

(defun sayShowWarning ()
  ($sayBrightly|
    "Warning: this is a temporary file and will be deleted the next")
  ($sayBrightly|
    " time you use )show. Rename it and FILE if you wish to")
  ($sayBrightly|
    " save the contents.")
  ($sayBrightly|
    ""))

—
25.53  )spool Command

spool man page

--- spool.help ---

====================================================================
A.23.  )spool
====================================================================

User Level Required: interpreter

Command Syntax:

- )spool [fileName]
- )spool

Command Description:

This command is used to save (spool) all AXIOM input and output into a file, called a spool file. You can only have one spool file active at a time. To start spool, issue this command with a filename. For example,

)spool integrate.out

To stop spooling, issue )spool with no filename.

If the filename is qualified with a directory, then the output will be placed in that directory. If no directory information is given, the spool file will be placed in the current directory. The current directory is the directory from which you started AXIOM or is the directory you specified using the )cd command.

Also See:
o )cd

---

18"cd" (25.11 p 764)
CHAPTER 25. SYSTEM COMMAND HANDLING

25.54 )summary Command

summary man page

| summary.help |

)credits : list the people who have contributed to Axiom

)help <command> gives more information

)quit : exit AXIOM

)abbreviation : query, set and remove abbreviations for constructors

)browse : start an Axiom http server on 127.0.0.1 port 8085

)cd : set working directory

)clear : remove declarations, definitions or values

)close : throw away an interpreter client and workspace

)compile : invoke constructor compiler

)copyright : show copyright and trademark information

)describe : show database information for a category, domain, or package

)display : display Library operations and objects in your workspace

)edit : edit a file

)fin : drop into lisp, use (restart) to return to the session

)frame : manage interpreter workspaces

)history : manage aspects of interactive session

)include : insert a file into a .input file

)library : introduce new constructors

)license : display the Axiom license file

)lisp : evaluate a LISP expression

)trace : trace functions

)pquit : ask if you really want to exit Axiom

)quit : exit Axiom

)read : execute AXIOM commands from a file

)regress : regression test an output spool file

)savesystem : save LISP image to a file

)set : view and set system variables

)show : show constructor information

)spool : log input and output to a file

)synonym : define an abbreviation for system commands

)system : issue shell commands

)tangle : extract chunks from a literate program to an input file

)trace : trace execution of functions

)trademark : declare that Axiom is a trademark of this software effort

)undo : restore workspace to earlier state

)what : search for various things by name
defun summary

[obey p??]
[concat p1197]
[getenv p255]

— defun summary —

(defun summary (l)
  (declare (ignore l))
  (obey (concat "cat " (getenv "AXIOM") "/doc/spadhelp/summary.help")))
25.55 )synonym Command

synonym man page

— synonym.help —

====================================================================
A.24. )synonym
====================================================================

User Level Required: interpreter

Command Syntax:

- )synonym
- )synonym synonym fullCommand
- )what synonyms

Command Description:

This command is used to create short synonyms for system command expressions. For example, the following synonyms might simplify commands you often use.

)synonym save history )save
)synonym restore history )restore
)synonym mail system mail
)synonym ls system ls
)synonym fortran set output fortran

Once defined, synonyms can be used in place of the longer command expressions. Thus

)fortran on

is the same as the longer

)set fortran output on

To list all defined synonyms, issue either of

)synonyms
)what synonyms

To list, say, all synonyms that contain the substring ‘‘ap’’, issue

)what synonyms ap

Also See:
defun The \texttt{)synonym} command

\begin{verbatim}
(defun \texttt{synonym} ((\&rest ignore)
  (declare (ignore ignore))
  (\texttt{synonymSpad2Cmd}))
\end{verbatim}

defun The \texttt{)synonym} command implementation

\begin{verbatim}
(defun \texttt{synonymSpad2Cmd} ()
  (let (line pair)
    (declare (special \$\texttt{CommandSynonymAlist}))
    (setq line (\texttt{getSystemCommandLine}))
    (if (string= line "")
      (\texttt{printSynonyms} nil)
      (progn
        (setq pair (\texttt{processSynonymLine} line))
        (if \$\texttt{CommandSynonymAlist}
          (putalist \$\texttt{CommandSynonymAlist} (car pair) (cdr pair))
          (setq \$\texttt{CommandSynonymAlist} (cons pair nil))))
    (\texttt{terminateSystemCommand}))
\end{verbatim}

19 “set” (25.51 p 1013) “what” (25.62 p 1091)
defun Return a sublist of applicable synonyms

The argument is a list of synonyms, and this returns a sublist of applicable synonyms at the current user level. [string2id-n p??] [selectOptionLC p751] [commandsForUserLevel p720] [$systemCommands p715] [$UserLevel p1012]

— defun synonymsForUserLevel —

(defun |synonymsForUserLevel| (arg)
  (let (cmd nl)
    (declare (special |$systemCommands| |$UserLevel|))
    (if (eq |$UserLevel| '|development|)
      arg
      (dolist (syn (reverse arg))
        (setq cmd (string2id-n (cdr syn) 1))
        (when (|selectOptionLC| cmd (|commandsForUserLevel| |$systemCommands|) nil)
          (push syn nl)))
    nl))

——

defun Get the system command from the input line

[strpos p1196] [substring p256] [$currentLine p??]

— defun getSystemCommandLine —

(defun |getSystemCommandLine| ()
  (let (p line)
    (declare (special |$currentLine|))
    (setq p (strpos "\" |$currentLine| 0 nil))
    (if p
      (setq line (substring |$currentLine| p nil))
      (setq line |$currentLine|))
    (string-left-trim "(#\space) line)))

——
defun Remove system keyword

[dropLeadingBlanks p??]
[maxindex p??]

— defun processSynonymLine,removeKeyFromLine —

(defun |processSynonymLine,removeKeyFromLine| (line)
  (prog (mx)
    (return
      (seq
        (setq line (|dropLeadingBlanks| line))
        (setq mx (maxindex line))
        (exit
          (do ((i 0 (1+ i)))
              ((> i mx) nil)
            (seq
              (exit
                (if (char= (elt line i) #\space)
                  (exit
                    (return
                      (do ((j (1+ i) (1+ j)))
                          ((> j mx) nil)
                        (seq
                          (exit
                            (if (char\= (elt line j) #\space)
                              (exit
                                (return
                                  (substring line j nil)))))))))))))))))

——

defun processSynonymLine

[processSynonymLine,removeKeyFromLine p1039]

— defun processSynonymLine —

(defun |processSynonymLine| (line)
  (cons
    (string2id-n line 1)
    (|processSynonymLine,removeKeyFromLine| line)))

——

This command is in the list of `noParseCommands 25.1` which means that its arguments are
passed verbatim. This will eventually result in a call to the function `handleNoParseCommands`.
25.56  )system Command

system man page

— system.help —

====================================================================
A.25. )system
====================================================================

User Level Required: interpreter

Command Syntax:

- )system cmdExpression

Command Description:

This command may be used to issue commands to the operating system while remaining in AXIOM. The cmdExpression is passed to the operating system for execution.

To get an operating system shell, issue, for example, )system sh. When you enter the key combination, Ctrl-D (pressing and holding the Ctrl key and then pressing the D key) the shell will terminate and you will return to AXIOM. We do not recommend this way of creating a shell because Lisp may field some interrupts instead of the shell. If possible, use a shell running in another window.

If you execute programs that misbehave you may not be able to return to AXIOM. If this happens, you may have no other choice than to restart AXIOM and restore the environment via )history )restore, if possible.

Also See:
o )boot
o )fin
o )lisp
o )pquit
o )quit

This command is in the list of $noParseCommands 25.1 which means that its arguments are passed verbatim. This will eventually result in a call to the function handleNoParseCommands 25.2

20  “boot” (25.4 p 758) “fin” (25.20 p 808) “lisp” (25.27 p 867) “pquit” (25.29 p 870) “quit” (25.30 p 872)
25.57 \( \text{)} \text{tangle Command} \)

\text{tangle man page}

--- tanglehelp ---

====================================================================
A.19. \text{)} tangle
====================================================================

User Level Required: interpreter

Command Syntax:

- \text{)} tangle [fileName]

Command Description:

This command is used to tangle pamphlet files.

\text{)} tangle matrix.input.pamphlet

will tangle the contents of the file matrix.input.pamphlet into
matrix.input. The ‘‘.input.pamphlet’’ is optional.

---

--- defun tangle ---

(defun \text{)} tangle (arg)
(let ([\$InteractiveMode]\ namestring dot1 dot2 outfile
    (chunkname "*") (extension "input"))
  (declare (special \$InteractiveMode\ \$options\)))
  (setq \$InteractiveMode\ t)
  (setq namestring (symbol-name (car arg)))
  (setq dot1 (position #\. namestring))
  (if dot1
    (setq outfile
      (concatenate 'string (subseq namestring 0 dot1) "." extension))
     (setq outfile
      (concatenate 'string (subseq namestring 0) "." extension)))
    (setq dot2 (position #\. namestring :from-end t))
    (cond
      ((and (numberp dot1) (numberp dot2) (< dot1 dot2)))
      ((and (numberp dot1) (numberp dot2) (= dot1 dot2)))
      (setq namestring (concatenate 'string namestring ".pamphlet"))))
(tangle namestring (concatenate 'string namestring "input.pamphlet")))
(if (probe-file namestring)
  (progn
    (tangle namestring chunkname outfile)
    (format t (concatenate 'string outfile " created from " namestring "~%")))
  (format t (concatenate 'string namestring " file not found~%"))))
25.58  )trademark Command

trademark man page

— trademark.help —

============================================
A.15. )trademark
============================================

Command Syntax:

- )trademark

Command Description:

This command displays the Axiom trademark information.

Also See:
- )license

———
25.59  )trace Command

trace man page

— trace.help —

====================================================================
A.26. )trace
====================================================================

User Level Required: interpreter

Command Syntax:

- )trace
- )trace )off

- )trace function [options]
- )trace constructor [options]
- )trace domainOrPackage [options]

where options can be one or more of

- )after S-expression
- )before S-expression
- )break after
- )break before
- )cond S-expression
- )count
- )count n
- )depth n
- )local op1 [... opN]
- )nonquietly
- )nt
- )off
- )only listOfDataToDisplay
- )ops
- )ops op1 [... opN ]
- )restore
- )stats
- )stats reset
- )timer
- )varbreak
- )varbreak var1 [... varN ]
- )vars
- )vars var1 [... varN ]
- )within executingFunction
CHAPTER 25. SYSTEM COMMAND HANDLING

Command Description:

This command is used to trace the execution of functions that make up the AXIOM system, functions defined by users, and functions from the system library. Almost all options are available for each type of function but exceptions will be noted below.

To list all functions, constructors, domains and packages that are traced, simply issue

)trace

To untrace everything that is traced, issue

)trace )off

When a function is traced, the default system action is to display the arguments to the function and the return value when the function is exited. Note that if a function is left via an action such as a THROW, no return value will be displayed. Also, optimization of tail recursion may decrease the number of times a function is actually invoked and so may cause less trace information to be displayed. Other information can be displayed or collected when a function is traced and this is controlled by the various options. Most options will be of interest only to AXIOM system developers. If a domain or package is traced, the default action is to trace all functions exported.

Individual interpreter, lisp or boot functions can be traced by listing their names after )trace. Any options that are present must follow the functions to be traced.

)trace f

traces the function f. To untrace f, issue

)trace f )off

Note that if a function name contains a special character, it will be necessary to escape the character with an underscore

)trace _/D_,1

To trace all domains or packages that are or will be created from a particular constructor, give the constructor name or abbreviation after

)trace.

)trace MATRIX
)trace List Integer

The first command traces all domains currently instantiated with Matrix. If
additional domains are instantiated with this constructor (for example, if you have used Matrix(Integer) and Matrix(Float)), they will be automatically traced. The second command traces List(Integer). It is possible to trace individual functions in a domain or package. See the )ops option below.

The following are the general options for the )trace command.

)break after
causes a Lisp break loop to be entered after exiting the traced function.

)break before
causes a Lisp break loop to be entered before entering the traced function.

)break
is the same as )break before.

)count
causes the system to keep a count of the number of times the traced function is entered. The total can be displayed with )trace )stats and cleared with )trace )stats reset.

)count n
causes information about the traced function to be displayed for the first n executions. After the nth execution, the function is untraced.

)depth n
causes trace information to be shown for only n levels of recursion of the traced function. The command

)trace fib )depth 10

will cause the display of only 10 levels of trace information for the recursive execution of a user function fib.

)math
causes the function arguments and return value to be displayed in the AXIOM monospace two-dimensional math format.

)nonquietly
causes the display of additional messages when a function is traced.

)nt
This suppresses all normal trace information. This option is useful if the )count or )timer options are used and you are interested in the statistics but not the function calling information.

)off
causes untracing of all or specific functions. Without an argument, all functions, constructors, domains and packages are untraced. Otherwise,
the given functions and other objects are untraced. To immediately retrace the untraced functions, issue \texttt{trace \ restore}.

\texttt{)only list0fDataToDisplay}
causes only specific trace information to be shown. The items are listed by using the following abbreviations:

- \texttt{a} display all arguments
- \texttt{v} display return value
- \texttt{1} display first argument
- \texttt{2} display second argument
- \texttt{15} display the 15th argument, and so on

\texttt{)restore}
causes the last untraced functions to be retraced. If additional options are present, they are added to those previously in effect.

\texttt{)stats}
causes the display of statistics collected by the use of the \texttt{)count} and \texttt{)timer} options.

\texttt{)stats reset}
resets to 0 the statistics collected by the use of the \texttt{)count} and \texttt{)timer} options.

\texttt{)timer}
causes the system to keep a count of execution times for the traced function. The total can be displayed with \texttt{)trace \ stats} and cleared with \texttt{)trace \ stats reset}.

\texttt{)varbreak var1 [... varN]}
causes a Lisp break loop to be entered after the assignment to any of the listed variables in the traced function.

\texttt{)vars}
causes the display of the value of any variable after it is assigned in the traced function. Note that library code must have been compiled (see description of command \texttt{)compile}) using the \texttt{)vartrace} option in order to support this option.

\texttt{)vars var1 [... varN]}
causes the display of the value of any of the specified variables after they are assigned in the traced function. Note that library code must have been compiled (see description of command \texttt{)compile}) using the \texttt{)vartrace} option in order to support this option.

\texttt{)within executingFunction}
causes the display of trace information only if the traced function is called when the given \texttt{executingFunction} is running.
The following are the options for tracing constructors, domains and packages.

```lisp
)local [op1 [... opN] ]
```
causes local functions of the constructor to be traced. Note that to
untrace an individual local function, you must use the fully qualified
internal name, using the escape character _ before the semicolon.

```lisp
)trace FRAC )local
)trace FRAC_;cancelGcd )off
```

```lisp
)ops op1 [... opN]
```
By default, all operations from a domain or package are traced when the
domain or package is traced. This option allows you to specify that only
particular operations should be traced. The command

```lisp
)trace Integer )ops min max _+ _-
```
traces four operations from the domain Integer. Since + and - are special
characters, it is necessary to escape them with an underscore.

Also See:
o )boot
o )lisp
o )ltrace

---

### The trace global variables

This decides when to give trace and untrace messages.

```lisp
defvar $traceNoisely
---

initvars

(defvar |$traceNoisely| nil)

---

[^21]: "boot" (25.4 p 758) "lisp" (25.27 p 867) "lltrace" (25.28 p 868)
\begin{verbatim}
defvar $reportSpadtrace
This reports the traced functions
   — initvars —
(defvar |$reportSpadtrace| nil)

defvar $optionAlist
   — initvars —
(defvar |$optionAlist| nil)

defvar $tracedMapSignatures
   — initvars —
(defvar |$tracedMapSignatures| nil)

defvar $traceOptionList
   — initvars —
(defvar |$traceOptionList| ''(|after| |before| |break| |cond| |count| |depth| |local| |mathprint|
   |nonquietly| |nt| |of| |only| |ops| |restore| |timer| |varbreak|
   |vars| |within|))
\end{verbatim}
defun resetTimers
    (defun |resetTimers| ()
        (declare (special /timerlist))
        (dolist (timer /timerlist)
            (set (intern (concat timer ",TIMER")) 0)))

defun resetSpacers
    (defun |resetSpacers| ()
        (declare (special /spacelist))
        (dolist (spacer /spacelist)
            (set (intern (concat spacer ",SPACE")) 0)))

defun resetCounters
    (defun |resetCounters| ()
        (declare (special /countlist))
        (dolist (k /countlist)
            (set (intern (concat k ",COUNT")) 0)))
defun ptimers

[sayBrightly p??]
[bright p??]
[quotient p??]
[concat p1197]
[float p??]
[/timerlist p??]

— defun ptimers —

(defun |ptimers| ()
(declare (special /timerlist |$timerTicksPerSecond|))
(if (null /timerlist)
  (|sayBrightly| " no functions are timed")
  (dolist (timer /timerlist)
    (|sayBrightly|`(" " ,@(|bright| timer) |:| " "
                  ,(quotient (eval (intern (concat timer ",TIMER")))
                  (|float| |$timerTicksPerSecond|)) " sec."))))

——

defun pspacers

[sayBrightly p??]
[bright p??]
[concat p1197]
[/spacelist p??]

— defun pspacers —

(defun |pspacers| ()
(declare (special /spacelist))
(if (null /spacelist)
  (|sayBrightly| " no functions have space monitored")
  (dolist (spacer /spacelist)
    (|sayBrightly|`
      " " ,@(|bright| spacer) |:| |
      ,(eval (intern (concat spacer ",SPACE"))) " bytes"))))

——
defun pcounters

[sayBrightly p??]
[bright p??]
[concat p1197]
[/countlist p??]

— defun pcounters —

(defun pcounters ()
  (declare (special /countlist))
  (if (null /countlist)
      (sayBrightly " no functions are being counted")
    (dolist (k /countlist)
      (sayBrightly " " ,@((bright k) : " " ,(eval (intern (concat k ",COUNT")) " times")))))

— —

defun transOnlyOption

[transOnlyOption p1053]
[upcase p1206]
[StackTraceOptionError p1054]
[qcar p??]
[qcdr p??]

— defun transOnlyOption —

(defun transOnlyOption (arg)
  (let (y n)
    (when (and (consp arg) (progn (setq n (qcar arg)) (setq y (qcdr arg)) t))
      (cond ((integerp n) (cons n (|transOnlyOption| y)))
            ((member (setq n (upcase n)) '('v a c)) (cons n (|transOnlyOption| y)))
            (t (|StackTraceOptionError|
                (cons "%1 The )trace option )only does not permit %2 as a legal option." (list (list n)))
                 (|transOnlyOption| y)))))

——
defun stackTraceOptionError

[\$traceErrorStack p??]

— defun stackTraceOptionError —

(defun stackTraceOptionError (x)
 (declare (special \$traceErrorStack))
 (push x \$traceErrorStack)
 nil)

defun removeOption

— defun removeOption —

(defun removeOption (op options)
 (let (opt t0)
   (do ((t1 options (cdr t1)) (optentry nil))
     ((or (atom t1)
       (progn (setq optentry (car t1)) nil)
       (progn (progn (setq opt (car optentry)) optentry) nil))
     (nreverse0 t0))
   (when (not (equal opt op)) (setq t0 (cons optentry t0)))))

defun domainToGenvar

[unabbrevAndLoad p??]
[getdatabase p1156]
[opOf p??]
[genDomainTraceName p59]
[evalDomain p1075]
[$doNotAddEmptyModeIfTrue p??]

— defun domainToGenvar —

(defun domainToGenvar (arg)
 (let ([\$doNotAddEmptyModeIfTrue| y g]
       (declare (special \$doNotAddEmptyModeIfTrue)))
       (setq \$doNotAddEmptyModeIfTrue t))
(when
  (and (setq y (unabbrevAndLoad arg))
       (eq (getdatabase (opOf y) 'constructorKind) 'domain))
  (setq g (genDomainTraceName y))
  (setq g (evalDomain y))
  g)))

defun subTypes

[lassoc p??]
[seq p??]
[exit p??]
[subTypes p1055]

—— defun subTypes ——

(defun |subTypes| (|mm| |sublist|)
 (prog (s)
   (return
    (seq
     (cond
      ((atom |mm|)
       (cond ((setq s (lassoc |mm| |sublist|)) s) (t |mm|)))
      (t
       (prog (t0)
         (setq t0 nil)
         (return
          (do ((t1 |mm| (cdr t1)) (|m| nil))
              ((or (atom t1) (progn (setq |m| (car t1)) nil)) (nreverse0 t0))
            (seq
             (exit
              (setq t0 (cons (|subTypes| |m| |sublist|) t0)))))))))))))

———

defun isListOfIdentifiers

[seq p??]
[exit p??]
[identp p1197]

—— defun isListOfIdentifiers ——
(defun is-list-of-identifiers (arg)
  (prog ()
    (return
      (seq
        (prog (t0)
          (setq t0 t)
          (return
            (do ((t1 nil (null t0)) (t2 arg (cdr t2)) (x nil))
                ((or t1 (atom t2) (progn (setq x (car t2)) nil)) t0)
              (seq
                (exit
                  (setq t0 (and t0 (identp x))))))))))

defun is-list-of-identifiers-or-strings

[seq p??]
[exit p??]
[identp p1197]

— defun is-list-of-identifiers-or-strings —

(defun is-list-of-identifiers-or-strings (arg)
  (prog ()
    (return
      (seq
        (prog (t0)
          (setq t0 t)
          (return
            (do ((t1 nil (null t0)) (t2 arg (cdr t2)) (x nil))
                ((or t1 (atom t2) (progn (setq x (car t2)) nil)) t0)
              (seq
                (exit
                  (setq t0 (and t0 (or (identp x) (stringp x))))))))))

defun get-previous-map-sub-names

[get p??]
[exit p??]
[seq p??]
[$InteractiveFrame p23]
--- defun getPreviousMapSubNames ---

(defun |getPreviousMapSubNames| (|traceNames|))
(prog (lmm subs)
  (declare (special |$InteractiveFrame|))
  (return
   (seq
    (progn
     (setq subs nil)
     (seq
      (do ((t0 (assocleft (caar |$InteractiveFrame|)) (cdr t0))
           (mapname nil))
       ((or (atom t0) (progn (setq mapname (car t0)) nil)) nil)
      (seq
       (exit
        (cond
         ((setq lmm
            (|get| mapname '|localModemap| |$InteractiveFrame|))
          (exit
           (cond
            ((member (cadar lmm) |traceNames|)
             (exit
              (do ((t1 lmm (cdr t1)) (|mm| nil))
                ((or (atom t1) (progn (setq |mm| (car t1)) nil)) nil)
               (seq
                (exit
                 (setq subs
                  (cons (cons mapname (cadr |mm|)) subs)))))))))))))
     (exit subs))))))

---

defun lassocSub

[lassq p??]

--- defun lassocSub ---

(defun |lassocSub| (x subs)
  (let (y)
    (if (setq y (lassq x subs))
      y
      x)))

---
defun rassocSub

[rassoc p??]

    — defun rassocSub —

(defun |rassocSub| (x subs)
  (let (y)
    (if (setq y (|rassoc| x subs))
      y
      x)))

defun isUncompiledMap

[get p??]  
[$InteractiveFrame p23]

    — defun isUncompiledMap —

(defun |isUncompiledMap| (x)
  (let (y)
    (declare (special |$InteractiveFrame|))
    (when (setq y (|get| x '|value| |$InteractiveFrame|))
      (and
       (eq (caar y) 'map)
       (null (|get| x '|localModemap| |$InteractiveFrame|))))))

defun isInterpOnlyMap

[get p??]  
[$InteractiveFrame p23]

    — defun isInterpOnlyMap —

(defun |isInterpOnlyMap| (map)
  (let (x)
    (declare (special |$InteractiveFrame|))
    (when (setq x (|get| map '|localModemap| |$InteractiveFrame|))
      (eq (caaar x) '|interpOnly|))))
defun isSubForRedundantMapName

(let (mapname tail)
  (declare (special $mapSubNameAlist))
  (when (setq mapname (rassocSub subname $mapSubNameAlist))
    (when (setq tail (member (cons mapname subname) $mapSubNameAlist))
      (member mapname (cdr (assocleft tail))))))

defun untraceMapSubNames

(let (|$mapSubNameAlist| subs)
  (declare (special $mapSubNameAlist $lastUntraced))
  (if (null (setq $mapSubNameAlist (getPreviousMapSubNames |traceNames|)))
    nil
  (dolist (name (setq subs (assocright $mapSubNameAlist)))
    (when (member name |traceNames|
      (null (setq $lastUntraced (setdifference $lastUntraced subs))))))

  )
defun funfind,LAM

[setq p??]  
[SEQ p??]  
[isFunctor p??]  
[exit p??]

--- defun funfind,LAM ---

(defun |funfind,LAM| (functor opname)
  (prog (ops tmp1)
    (return
      (seq
        (progn
          (setq ops (|isFunctor| functor))
          (prog t0)
          (setq t0 nil)
          (return
            (do ((t1 ops (cdr t1)) (u nil))
                ((or (atom t1) (progn (setq u (car t1)) nil)) (nreverse0 t0))
              (seq
                (exit
                  (cond
                    ((and (consp u)
                        (progn
                          (setq tmp1 (qcar u))
                          (and (consp tmp1) (equal (qcar tmp1) opname)))
                          (setq t0 (cons u t0))))))))))))

---

defmacro funfind

--- defmacro funfind ---

(defmacro |funfind| (&whole t0 &rest notused &aux t1)
  (declare (ignore notused))
  (dsetq t1 t0)
  (cons '|funfind,LAM| (wrap (cdr t1) '(quote quote))))

---
defun isDomainOrPackage

[refvecp p??]
[poundsign p??]
isFunctor p??
opOf p??]

— defun isDomainOrPackage —

(defun isDomainOrPackage (dom)
  (and
   (refvecp dom)
   (> (!#* dom) 0)
   (isFunctor! (opOf! (elt dom 0))))

———

defun isTraceGensym

gensymp p??]

— defun isTraceGensym —

(defun isTraceGensym (x)
  (gensymp x))

———

defun flattenOperationAlist

[seq p??]
[exit p??]

— defun flattenOperationAlist —

(defun flattenOperationAlist! (opAlist)
  (prog (op mmList! res)
    (return
     (seq
      (progn
       (setq res nil)
       (do ((t0 opAlist (cdr t0)) (t1 nil))
         ((or (atom t0)
             (progn (setq t1 (car t0)) nil)))
       nil)))
    nil)))
(progn
  (progn (setq op (car t1)) (setq |mmList| (cdr t1)) t1)
  nil))

(seq
  (exit
    (setq |res|
      (append |res|
        (prog (t2)
          (setq t2 nil)
          (return
            (do ((t3 |mmList| (cdr t3)) (mm nil))
              ((or (atom t3)
                  (progn (setq mm (car t3)) nil)) (nreverse0 t2))
              (seq
                (exit
                  (setq t2 (cons (cons op mm) t2)))))))))))))))

(defun letPrint

  [lassoc p??]
  [isgenvar p1064]
  [isSharpVarWithNum p1063]
  [gensym p??]
  [sayBrightlyNT p??]
  [bright p??]
  [shortenForPrinting p1067]
  [hasPair p1067]
  [pname p1195]
  [break p87]
  [$letAssoc p??]

  — defun letPrint —

(defun |letPrint| (x |val| |currentFunction|)
  (prog (y)
    (declare (special|$letAssoc$)))
    (return
      (progn
        (cond ((and|$letAssoc$)
              (or
                (setq y (lassoc |currentFunction||$letAssoc$))
                (setq y (lassoc '|all||$letAssoc$)))))
        (cond
((and (or (eq y 'all))
  (member x y))
  (null
   (or (isgenvar x) (isSharpVarWithNum x) (gensymp x))))
  (sayBrightlyNT (append (|bright| x) (cons '[: | nil])))
  (prin1 (|shortenForPrinting| |val|))
  (terpri)))
  (cond
    ((and (setq y (|hasPair| 'break y))
     (or (eq y 'all))
     (and (member x y)
      (null (member (elt (pname x) 0) '($ |#|)))
      (null (gensymp x))))
    (break|
     (append
      (|bright| |currentFunction|)
      (cons "breaks after"
      (append
       (|bright| x)
       (cons ":= " (cons (|shortenForPrinting| |val|) nil)))))))
    (t nil))))
  |val|)))

———

defun Identifier beginning with a sharpsign-number?

This tests if x is an identifier beginning with # followed by a number.

[isSharpVar p1064]
[pname p1195]
[qsize p??]
[ digitp p1195]
[dig2fix p??]

— defun isSharpVarWithNum —

(defun |isSharpVarWithNum| (x)
  (let (p n d ok c)
    (cond
      ((null (|isSharpVar| x)) nil)
      ( (> 2 (setq n (qsize (setq p (pname x)))))) nil)
      (t
       (setq ok t)
       (setq c 0)
       (do ((t1 (1- n)) (i 1 (1+ i)))
         ((or (> i t1) (null ok)) nil)
         (setq d (elt p i)))

}}-
(when (setq ok (digitp d))
  (setq c (+ (* 10 c) (dig2fix d))))
(when ok c))))

---

defun Identifier beginning with a sharpsign?

This tests if x is an identifier beginning with #

[identp p1197]

---

---

defun isgenvar

[isgenvar p1064]
[lassoc p??]
[isgenvar p1064]
[isSharpVarWithNum p1063]
[gensymp p??]
[mathprint p??]
[print p??]
(defun letPrint2 (x printform currentFunction)
(prog ($BreakMode $flag) y)
declare (special $BreakMode $letAssoc))
return
(progn
(setq $BreakMode nil)
(cond
((and $letAssoc)
 (or (setq y (lassoc currentFunction $letAssoc)))
 (setq y (lassoc 'all $letAssoc))))
(cond
((and
 (or (eq y 'all) (member x y))
 (null (or (isgenvar x) (lisSharpVarWithNum x) (gensymp x))))
(setq $BreakMode 'letPrint2)
(setq $flag nil)
(catch 'letPrint2
 (mathprint (cons '=' (cons x (cons printform nil)))) $flag)
(cond
((eq $flag 'letPrint2) (print printform))
(t nil)))))
(cond
((and
 (setq y (hasPair 'break y))
 (or (eq y 'all))
 (and
 (member x y)
 (null (member (elt (pname x) 0) ('$ #')))
 (null (gensymp x))))
(break)
append
(bright currentFunction)
(cons "breaks after"
 (append [: (bright x) (cons ':= | (cons printform nil))])
 (t nil)))
x)))}
defun letPrint3

This is the version for use when we have our hands on a function to convert the data into type "Expression" |letPrint2 p1064|

|lassoc p??|  
isgenvar p1064|  
isSharpVarWithNum p1063|  
gensymp p??|  
mathprint p??|  
spadcall p??|  
print p??|  
hasPair p1067|  
pname p1195|  
break p87|  
bright p??|  
$BreakMode p901|  
$letAssoc p??|

| defun letPrint3 |

| (defun |letPrint3| (x |xval| |printfn| |currentFunction|)  
(prog ($BreakMode |flag| y)  
(declare (special $BreakMode |$letAssoc|))  
(return  
(progn  
(setq $BreakMode nil)  
(cond  
((and $letAssoc  
(or (setq y (lassoc |currentFunction| $letAssoc)))  
(setq y (lassoc '|all| $letAssoc)))  
(cond  
((and  
(or (eq y '|all|) (member x y))  
(null (or (isgenvar x) (|isSharpVarWithNum| x) (gensymp x))))  
(setq $BreakMode 'letPrint2)  
(setq |flag| nil)  
(catch 'letPrint2  
(mathprint  
(cons '=' (cons x (cons (spadcall |xval| |printfn|) nil)))  
|flag|)  
(cond  
((eq |flag| 'letPrint2) (|print| |xval|))  
(t nil))))  
)  
))  
)  
))  
}  
--- defun letPrint3 ---  

| (defun |letPrint3| (x |xval| |printfn| |currentFunction|)  
(prog ($BreakMode |flag| y)  
(declare (special $BreakMode |$letAssoc|))  
(return  
(progn  
(setq $BreakMode nil)  
(cond  
((and $letAssoc  
(or (setq y (lassoc |currentFunction| $letAssoc)))  
(setq y (lassoc '|all| $letAssoc)))  
(cond  
((and  
(or (eq y '|all|) (member x y))  
(null (or (isgenvar x) (|isSharpVarWithNum| x) (gensymp x))))  
(setq $BreakMode 'letPrint2)  
(setq |flag| nil)  
(catch 'letPrint2  
(mathprint  
(cons '=' (cons x (cons (spadcall |xval| |printfn|) nil)))  
|flag|)  
(cond  
((eq |flag| 'letPrint2) (|print| |xval|))  
(t nil))))  
)  
))  
)  
)
(and
  (member x y)
  (null (member (elt (pname x) 0) '($ |#|)))
  (null (gensym x))))
(|break|
  (append
   (|bright| |currentFunction|)
   (cons "breaks after"
     (append (|bright| x) (cons ":= " (cons |xval| nil)))))))
(x)))

----------
defun hasPair

[qcar p??]
[qcdr p??]
[hasPair p1067]

— defun hasPair —

(defun |hasPair| (key arg)
  (prog (tmp1 a)
    (return
      (cond
        (atom arg) nil
        ((and (consp arg)
          (progn
            (setq tmp1 (qcar arg))
            (and (consp tmp1)
              (equal (qcar tmp1) key)
              (progn (setq a (qcdr tmp1)) t))))
           a)
        (t (|hasPair| key (cdr arg)))))))

----------
defun shortenForPrinting

[isDomainOrPackage p1061]
[devaluate p??]

— defun shortenForPrinting —
(defun |shortenForPrinting| (|val|)
  (if (|isDomainOrPackage| |val|)
      (|devaluate| |val|)
      |val|))

---

defun getOption

[assoc p??]

---

defun orderBySlotNumber

[seq p??]
[assocright p??]
[orderList p??]
[exit p??]

---

(defun |orderBySlotNumber| (arg)
  (prog (n)
    (return
      (seq
        (assocright
          (|orderList|
            (prog (t0)
              (setq t0 nil)
              (return
                (do ((t1 arg (cdr t1)) (x nil))
                    ((or (atom t1)
                         (progn (setq x (car t1)) nil)
                         (progn (progn (setq n (caddr x)) x) nil))
                         (nreverse0 t0))
                        (nreverse0 t0))))
        (exit
          (setq t0 (cons (cons n x) t0))))))))))
defun spadReply,printName

(seq p??)
(qcar p??)
isDomainOrPackage p1061
(exit p??)
devaluate p??

— defun spadReply,printName —

(defun |spadReply,printName| (x)
(prog (|d|)
(return
(seq
(if (and (and (consp x) (progn (setq |d| (qcar x)) t))
    (|isDomainOrPackage| |d|))
    (exit (|devaluate| |d|)))
    (exit x))))))

————

defun spadReply

(seq p??)
(exit p??)
|spadReply,printName| p1069
[/tracenames p??]

— defun spadReply —

(defun |spadReply| ()
(prog ()
(declare (special /tracenames))
(return
(seq
(prog (t0)
(setq t0 nil)
(return
(do ((t1 /tracenames (cdr t1)) (x nil))
    ((or (atom t1) (progn (setq x (car t1)) nil)) (nreverse0 t0))
    (seq
      (exit
        (setq t0 (cons (|spadReply,printName| x) t0))))))))))

————
---

defun remover

(remover p1070)

---

— defun remover —

(defun remover (lst item)
  (cond
   ((null (consp lst)) (cond ((equal lst item) nil) (t lst)))
   ((equal (car lst) item) (cdr lst))
   (t
    (rplnode lst (remover (car lst) item) (remover (cdr lst) item))
    (rplaca lst (remover (car lst) item))
    (rplacd lst (remover (cdr lst) item))
    (rplaca lst (remover (cdr lst) item))
    lst)))

---

defvar $constructors

— initvars —

(defvar $constructors nil)

---

defun stupidIsSpadFunction

[strpos p1196]
[ pname p1195]

— defun stupidIsSpadFunction —

(stupidIsSpadFunction (fn)
 (strpos ";" (pname fn) 0 nil))

---
defun compileBoot

([/D,1 p??])

— defun compileBoot —

(defun |compileBoot| (fn)
  (|/D,1| (list fn) '(/comp) nil nil))

———
25.60  )undo Command

undo man page

— undo.help —

====================================================================
A.27.  )undo
====================================================================

User Level Required:  interpreter

Command Syntax:

- )undo
- )undo integer
- )undo integer [option]
- )undo )redo

where option is one of

- )after
- )before

Command Description:

This command is used to restore the state of the user environment to an
earlier point in the interactive session. The argument of an )undo is an
integer which must designate some step number in the interactive session.

)undo n
)undo n )after

These commands return the state of the interactive environment to that
immediately after step n. If n is a positive number, then n refers to step
number n. If n is a negative number, it refers to the nth previous command
(that is, undoes the effects of the last -n commands).

A )clear all resets the )undo facility. Otherwise, an )undo undoes the effect
of )clear with options properties, value, and mode, and that of a previous
undo. If any such system commands are given between steps n and n + 1 (n >
0), their effect is undone for )undo m for any 0 < m <= n .

The command )undo is equivalent to )undo -1 (it undoes the effect of the
previous user expression). The command )undo 0 undoes any of the above system
commands issued since the last user expression.

)undo n )before
This command returns the state of the interactive environment to that immediately before step n. Any \texttt{undo} or \texttt{clear} system commands given before step n will not be undone.

\texttt{undo \texttt{undo}}

This command reads the file \texttt{redo.input} created by the last \texttt{undo} command. This file consists of all user input lines, excluding those backtracked over due to a previous \texttt{undo}.

The command \texttt{\texttt{history \texttt{write}}} will eliminate the \texttt{‘\texttt{undo}’} command lines of your program.

Also See:
\texttt{o \texttt{\texttt{history}}}

\section*{25.61 Evaluation}

Some Antique Comments About the Interpreter

\texttt{EVAL} \texttt{BOOT} contains the top level interface to the Scratchpad-II interpreter. The Entry point into the interpreter from the parser is \texttt{processInteractive}.

The type analysis algorithm is contained in the file \texttt{BOTMUP} \texttt{BOOT}, and \texttt{MODSEL boot}, the map handling routines are in \texttt{MAP} \texttt{BOOT} and \texttt{NEWMAP} \texttt{BOOT}, and the interactive coerce routines are in \texttt{COERCE} \texttt{BOOT} and \texttt{COERCEF} \texttt{BOOT}.

\textbf{Conventions:} All spad values in the interpreter are passed around in triples. These are lists of three items:

\begin{verbatim}
[value,mode,environment]
\end{verbatim}

The value may be wrapped (this is a pair whose \texttt{CAR} is the atom \texttt{WRAPPED} and whose \texttt{CDR} is the value), which indicates that it is a real value, or unwrapped in which case it needs to be \texttt{EVA}Led to produce the proper value. The mode is the type of value, and should always be completely specified (not contain \texttt{$EmptyMode$}). The environment is always empty, and is included for historical reasons.

\textbf{Modemaps:} Modemaps are descriptions of compiled Spad function which the interpreter uses to perform type analysis. They consist of patterns of types for the arguments, and conditions the types must satisfy for the function to apply. For each function name there is a list of modemaps in file \texttt{modemap DATABASE} for each distinct function with that name.

\footnote{\texttt{“history” (25.23 p 821)}}
The following is the list of the modmaps for "*" (multiplication. The first modemap (the one with the labels) is for module multiplication which is multiplication of an element of a module by a member of its scalar domain.

This is the signature pattern for the modemap, it is of the form:

\[(\text{DomainOfComputation TargetType } \langle \text{ArgumentType ...} \rangle)\]

\[\text{This is the predicate that needs to be satisfied for the modemap to apply}\]

\[V\]

\[-------------/\]

\[(\langle *1 \ *1 \ *2 \ *1 \rangle)\]

\[\text{CATDEF} \rightleftarrows \text{This is the file where the function was defined}\]

\[(\langle *1 \ *1 \ *2 \ *1 \rangle)\]

\[\text{CATDEF}\]

\[(\langle *1 \ *1 \ *2 \ *1 \rangle)\]

\[\\]

\[\text{Environments: Environments associate properties with atoms.}\]

Some common properties are:

- **modeSet**: During interpretation we build a modeSet property for each node in the expression. This is (in theory) a list of all the types possible for the node. In the current implementation these modeSets always contain a single type.

- **value**: Value properties are always triples. This is where the values of variables are stored. We also build value properties for internal nodes during the bottom up phase.

- **mode**: This is the declared type of an identifier.

Frequently used global variables:

- **$genValue**: if true then evaluate generated code, otherwise leave code unevaluated. If $genValue is false then we are compiling.

- **$op**: name of the top level operator (unused except in map printing)

- **$mapList**: list of maps being type analyzed, used in recursive map type analysis.

- **$compilingLoop**: true when compiling a loop body, used to control nesting level of interp-only loop CATCH points
• $\textit{interpOnly}$: true when in interpret only mode, used to call alternate forms of \texttt{COLLECT} and \texttt{REPEAT}.

• $\textit{inCOLLECT}$: true when compiling a \texttt{COLLECT}, used only for hacked stream compiler.

• $\textit{StreamFrame}$: used in printing streams, it is the environment where local stream variables are stored

• $\textit{declaredMode}$: Weak type propagation for symbols, set in \texttt{upCOERCE} and \texttt{upLET}. This variable is used to determine the alternate polynomial types of Symbols.

• $\textit{localVars}$: list of local variables in a map body

• $\textit{MapArgumentTypeList}$: hack for stream compilation

defun evalDomain

[sayMSG p29]
[concat p1197]
[prefix2String p??]
[startTimingProcess p??]
[eval p??]
[mkEvalable p1075]
[stopTimingProcess p??]
[$evalDomain p1075]

— defun evalDomain —

(defun evalDomain (form)
(let (result)
(declare (special $evalDomain))
(when $evalDomain
  (sayMSG
    (concat " instantiating" (prefix2String form)))))
(startTimingProcess '|instantiation|)
(setq result (eval (mkEvalable form))
(stopTimingProcess '|instantiation|)
result))

——

defun mkEvalable

[qcar p??]
[qcdr p??]
[mkEvalable p1075]
(defun mkEvalable (form)
  (let (op argl kind cosig)
    (declare (special (list $Integer $EmptyMode)))
    (cond
      ((consp form)
        (setq op (qcar form))
        (setq argl (qcdr form))
        (cond
          ((eq op 'quote) form)
          ((eq op 'wrapped) (mkEvalable (devaluate argl)))
          ((eq op '|Record|) (mkEvalableRecord form))
          ((eq op '|Union|) (mkEvalableUnion form))
          ((eq op '|Mapping|) (mkEvalableMapping form))
          ((eq op '|Enumeration|) form)
          (t
           (loadIfNecessary op))
        (setq kind (getdatabase op 'constructorkind))
        (cond
          ((setq cosig (getdatabase op 'cosig))
           (cons op
             (loop for x in argl for typeFlag in (rest cosig)
                collect
                (cond
                  (typeFlag
                    (cond
                      ((eq kind '|category|) (mkq x))
                      ((vecp x) (mkq x))
                      (t
                       (loadIfNecessary x)
                       (mkEvalable x))))
                    ((and (consp x) (eq (qcar x) 'quote)) x)
                    ((and (consp x) (eq (qcar x) '|#|) (consp (qcdr x))
                       (eq (qcdr (qcdr x)) nil))
                       (list 'size (mkq (qcar (qcdr x)))))))))))
    t))
  (defun devaluate (form)
    (let (op argl)
      (declare (special (list $Integer $EmptyMode)))
      (cond
        ((consp form)
         (setq op (qcar form))
         (setq argl (qcdr form))
         (cond
          ((eq op 'quote) form)
          ((eq op 'wrapped) (devaluate argl))
          ((eq op '|Record|) (mkEvalableRecord form))
          ((eq op '|Union|) (mkEvalableUnion form))
          ((eq op '|Mapping|) (mkEvalableMapping form))
          ((eq op '|Enumeration|) form)
          (t
           (loadIfNecessary op))))
    t)
25.61. EVALUATION

(t (mak x)))))
(t
(cons op
  (loop for x in argl
    collect (makEvalable x)))))
((equal form $EmptyModel$) $Integer$)
((and (identp form) (constructor? form)) (list form))
((fbpip form) (bpiname form))
(t form)))

defun mkEvalableUnion

[makEvalable p1075]
— defun mkEvalableUnion —

(defun |mkEvalableUnion| (form)
  (cond
    ((isTaggedUnion? form)
      (cons (car form)
        (loop for item in (rest form)
          collect (list ': (second item) (makEvalable (third item)))))
      (t
        (cons (car form)
          (loop for d in (rest form)
            collect (makEvalable d))))))

——

defun isTaggedUnion

— defun isTaggedUnion —

(defun |isTaggedUnion| (u)
  (and (eq (car u) 'Union) (eq (caadr u) ':))))
defun mkEvalableRecord

— defun mkEvalableRecord —

(defun mkEvalableRecord (form)
  (cons
   (car form)
   (loop for item in (rest form)
      collect (list (quote |:|) (second item) (mkEvalable (third item)))))))

——

defun mkEvalableMapping

— defun mkEvalableMapping —

(defun mkEvalableMapping (form)
  (cons
   (car form)
   (loop for d in (rest form)
      collect (mkEvalable d))))

——

defun evaluateType

Takes a parsed, unabbreviated type and evaluates it, replacing type valued variables with
their values, and calling bottomUp on non-type valued arguemnts to the constructor and
finally checking to see whether the type satisfies the conditions of its modemap.

—isDomainValuedVariable p1105
[qcar p??]
[qcdr p??]
[mkAtree p??]
[bottomUp p??]
[objVal p448]
[getValue p??]
[evaluateSignature p1083]
[member p1198]
[evaluateType p1078]
— defun evaluateType —

(defun evaluateType (form)
  (let ((|$expandSegments| domain formp op argl)
        (declare (special |$expandSegments| |$EmptyMode|)))
    (cond
      ((setq domain (|isDomainValuedVariable| form)) domain)
      ((equal form |$EmptyMode|) form)
      ((eq form '?) |$EmptyMode|)
      ((stringp form) form)
      ((eq form '$) form)
      (t
       (setq |$expandSegments| nil)
       (cond
         ((and (consp form) (eq (qcar form) '|typeOf|) (consp (qcdr form))
          (eq (qcdr (qcdr form)) nil))
          (setq formp (|mkAtree| form))
          (|bottomUp| formp)
          (|objVal| (|getValue| formp)))
         ((consp form)
          (setq op (qcar form))
          (setq argl (qcdr form))
          (cond
            ((eq op 'category)
             (cond
              ((consp argl)
               (cons op
                  (cons (qcar argl)
                    (loop for s in (qcdr argl)
                          collect (|evaluateSignature| s))))))
             (t form))
            ((|member| op '(|Join| |Mapping|))
             (cons op
               (loop for arg in argl
                 collect (|evaluateType| arg)))
             ((eq op '|Union|)
              (cond
                ((and argl (consp (car argl)) (consp (qcdr (car argl)))
                 (consp (qcdr (qcdr (car argl))))
                 (eq (qcdr (qcdr (qcdr (car argl)))) nil)
                 (|member| (qcar (car argl)) '((:: |Declare|)))
                 (cons op
                  (loop for item in argl
                    collect
                    ...))
                (t form))
            (t form)))
         (t form)))))}
(list '|:| (second item) (|evaluateType| (third item)))))))
(t
  (cons op
    (loop for arg in argl
      collect (|evaluateType| arg))))))
((eq op '|Record|)
 (cons op
  (loop for item in argl
    collect
    (list '|:| (second item) (|evaluateType| (third item)))))))
((eq op '|Enumeration|) form)
  (t (|evaluateType1| form)))
((|constructor?| form)
  (if (atom form)
    (|evaluateType| (list form))
    (|throwEvalTypeMsg|
      (format nil
        "Although %1 is the name of a constructor, a full type must be ~
         specified in the context you have used it. Issue )show %2 ~
         for more information."))
    (list form)))
  (t (|throwEvalTypeMsg| "%1p is not a valid type." (list form)))))

defun Evaluate args passed to a constructor

Evaluates the arguments passed to a constructor.

[constructor? p??]
[getConstructorSignature p??]
[throwEvalTypeMsg p1082]
[replaceSharps p1104]
[categoryForm? p??]
[evaluateType p1078]
[evalCategory p1105]
[getdatabase p1156]
[mkAtree p??]
[putTarget p??]
[bottomUp p??]
[qcar p??]
[qcdr p??]
[getAndEvalConstructorArgument p1104]
[coerceOrRetract p703]
[objValUnwrap p48]
[throwKeyedMsgCannotCoerceWithValue p??]
[makeOrdinal p1083]
25.61. EVALUATION

— defun evaluateType1 —

(defun evaluateType1 (form)
  (let (op argl sig m1 xp tree tmp1 m1 z1 zt zv v typeList (argnum 0))
    (declare (special |$quadSymbol| |$EmptyMode|))
    (setq op (car form))
    (setq argl (cdr form))
    (cond
      ((constructor? op)
       (cond
        (null (setq sig (|getConstructorSignature| form)))
        (|throwEvalTypeMsg|
          "You cannot now use %1p in the context you have it." (list form))
        (t
         (setq m1 (cdr sig))
         (setq m1 (|replaceSharps| m1 form))
         (cond
          ((not (eql (|#| argl) (|#| m1)))
           (|throwEvalTypeMsg|
             (format nil
               "Although %1 is the name of a constructor, a full type must be ~
                 specified in the context you have used it. Issue \show %2 ~
                 for more information."
               )
             )
           )
          (t
           (loop for x in argl for m in m1
             do
             (setq typeList
               (cons
                (cond
                 ((categoryForm? m)
                  (setq m (|evaluateType| (subst x '$ m)))
                  (if (evalCategory xp (|evaluateType| x)) m)
                  xp
                  (|throwEvalTypeMsg| "%1p is not a valid type."
                    )
                (t
                 (setq m (|evaluateType| m))
                 )
                 (cond
                  ((and (eq (getdatabase (|opOf| m) 'constructorkind) '|domain|)
                    (setq tree (|mkAtree| x))
                    (|putTarget| tree m)
                    )
                  (progn
                   (setq tmp1 (|bottomUp| tree))
                   (and (consp tmp1)
                     (eq (qcdr tmp1) nil)))
                  )
                  (setq m1 (qcar tmp1))
                )))
             )))
           )))
        )))
    )))
(setq z1 (getAndEvalConstructorArgument tree))
(setq zt (car z1))
(setq zv (cdr z1))
(if (setq v (coerceOrRetract z1 m))
  (objValUnwrap v)
  (throwKeyedMsgCannotCoerceWithValue zv zt m)))
(t
  (when (equal x $EmptyMode) (setq x $quadSymbol))
  (throwEvalTypeMsg
    "Cannot convert the %1 argument of %3p to the type %2p ."
    (list (makeOrdinal (incf argnum)) m form))))))
(cons op (nreverse typeList))))
(t (throwEvalTypeMsg
    "Category, domain or package constructor %1 is not available."
    (list op))))))

defvar $noEvalTypeMsg

— initvars —

(defvar $noEvalTypeMsg nil)

defun throwEvalTypeMsg

[spadThrow p??]
[throwKeyedMsg p??]
[$noEvalTypeMsg p1082]

— defun throwEvalTypeMsg —

(defun |throwEvalTypeMsg| (msg args)
  (declare (special |$noEvalTypeMsg|))
  (if |$noEvalTypeMsg|
      (spadThrow)
      (throwKeyedMsg msg args)))
defun makeOrdinal

— defun makeOrdinal —

(defun makeOrdinal (i)
  (elt '(first second third fourth fifth sixth seventh eighth ninth tenth)
       (1- i)))

defun evaluateSignature

Calls evaluateType on a signature.

[evaluateType p1078]

— defun evaluateSignature —

(defun evaluateSignature (sig)
  (cond
   ((and (consp sig) (eq (qcar sig) 'signature) (consp (qcdr sig))
     (consp (qcdr (qcdr sig))) (eq (qcdr (qcdr (qcdr sig))) nil))
    (cons 'signature (cons (qcar (qcdr sig))
      (list
       (loop for z in (qcar (qcdr (qcdr sig)))
         collect (if (eq z '$) z (|evaluateType| z))))))
    (t sig)))

defun recordFrame

[diffAlist p1085]
[seq p??]
[exit p??]
[frameRecord p35]
[InteractiveFrame p23]
[previousBindings p36]

— defun recordFrame —

(defun recordFrame (systemNormal)
  (prog (currentAlist delta)
(declare (special |$frameRecord| |$InteractiveFrame| |$previousBindings|))
(return
  (seq
    (setq currentAlist (ifcar |$frameRecord|))
    (setq delta (|diffAlist| (caar |$InteractiveFrame|) |$previousBindings|))
    (cond
      ((eq systemNormal 'system)
       (cond
         ((null delta)
          (return nil))
         (t
          (setq delta (cons '|systemCommand| delta))))))
    (setq |$frameRecord| (cons delta |$frameRecord|))
    ; copy all but the individual properties
    ; note that this loop makes no sense. In boot it read:
    ; [cons(first x, [cons(first y,rest y) for y in rest x]) for x
    ; in caar $InteractiveFrame
    ; ... but cons(first y, rest y) == y
    (setq |$previousBindings|)
    (prog (tmp0)
      (setq tmp0 nil)
      (return
        (do ((tmp1 (caar |$InteractiveFrame|) (cdr tmp1)) (x nil))
            ((or (atom tmp1)
              (progn (setq x (car tmp1)) nil))
             (nreverse0 tmp0))
        (seq
          (exit
           (setq tmp0
             (cons
              (cons
               (car x)
               (prog (tmp2)
                 (setq tmp2 nil)
                 (return
                  (do ((tmp3 (cdr x) (cdr tmp3)) (y nil))
                      ((or (atom tmp3)
                        (progn (setq y (car tmp3)) nil))
                         (nreverse0 tmp2))
                      (seq
                         (exit
                          (setq tmp2 (cons (cons (car y) (cdr y)) tmp2)))))))
             tmp0)))))
        (first |$frameRecord|))))
defun diffAlist

diffAlist(new, old) ==
-- record only those properties which are different
for (pair := [name,:proplist]) in new repeat
  -- name has an entry both in new and old world
  -- (1) if the old world had no proplist for that variable, then
  --     record NIL as the value of each new property
  -- (2) if the old world does have a proplist for that variable, then
  --     a) for each property with a value: give the old value
  --     b) for each property missing: give NIL as the old value
  oldPair := ASSQ(name, old) =>
    null (oldProplist := CDR oldPair) =>
      -- record old values of new properties as NIL
      acc := [ [name,:[ [prop] for [prop,:.] in proplist ]],:acc]
    deltas := nil
    for (propval := [prop,:val]) in proplist repeat
      null (oldPropval := ASSOC(prop, oldProplist)) =>
        -- missing property
        deltas := [ [prop],:deltas]
      EQ(CDR oldPropval, val) => 'skip
      deltas := [oldPropval,:deltas]
    deltas => acc := [ [name,:NREVERSE deltas],:acc]
  acc := [ [name,:[ [prop] for [prop,:.] in proplist ]],:acc]
-- record properties absent on new list (say, from a cl all)
for (oldPair := [name,:r]) in old repeat
  r and null LASSQ(name, new) =>
    acc := [oldPair,:acc]
  -- name has an entry both in new and old world
  -- (1) if the new world has no proplist for that variable
  -- (a) if the old world does, record the old proplist
  -- (b) if the old world does not, record nothing
  -- (2) if the new world has a proplist for that variable, it has
  --     been handled by the first loop.
  res := NREVERSE acc
  if (BOUNDP '$reportundo and $reportundo) then reportUndo res
  res

[assq p1200]
[tmp1 p?]
[seq p??]
[exit p??]
[assoc p??]
[llassq p??]
[reportUndo p43]

— defun diffAlist —

(defun |diffAlist| (new old)
  (prog (proplist oldPair oldProplist val oldPropval deltas prop name r acc res)
(return
(seq
(progn
(do ((tmp0 new (cdr tmp0)) (pair nil))
  ((or (atom tmp0)
       (progn (setq pair (car tmp0)) nil)
       (progn
         (progn
           (setq name (car pair))
           (setq proplist (cdr pair))
           pair)
         nil))
  nil)
 nil)
(seq
(exit
(cond
  ((setq oldPair (assq name old))
   (cond
     ((null (setq oldProplist (cdr oldPair)))
      (setq acc
        (cons
          (cons
            name
            (prog (tmp1)
              (setq tmp1 nil)
              (return
               (do ((tmp2 proplist (cdr tmp2)) (tmp3 nil))
                 ((or (atom tmp2)
                    (progn (setq tmp3 (car tmp2)) nil)
                    (progn
                      (progn
                        (setq prop (car tmp3))
                        tmp3)
                    nil))
                 (nreverse0 tmp1))
               (seq
                (exit
                 (setq tmp1 (cons (cons prop nil) tmp1)))))))))
     (t
      (setq deltas nil)
      (do ((tmp4 proplist (cdr tmp4)) (|propval| nil))
        ((or (atom tmp4)
             (progn (setq |propval| (car tmp4)) nil)
             (progn
              (progn
                (setq prop (car |propval|))
                (setq val (cdr |propval|))
                |propval|)
              nil))
        nil)
      (seq

(exit
 (cond
   ((null (setq oldPropval (assoc prop oldProplist)))
    (setq deltas (cons (cons prop nil) deltas)))
   ((eq (cdr oldPropval) val) '|skip|)
   (t (setq deltas (cons oldPropval deltas)))))))
 (when deltas
  (setq acc
    (cons (cons name (nreverse deltas)) acc))))))
(t
 (setq acc
  (cons
   (cons
    name
    (prog (tmp5)
      (setq tmp5 nil)
      (return
       (do ((tmp6 proplist (cdr tmp6)) (tmp7 nil))
         ((or (atom tmp6)
          (progn (setq tmp7 (CAR tmp6)) nil)
          (progn
            (setq prop (CAR tmp7))
            tmp7)
         nil))
        (nreverse0 tmp5))
      (seq
       (exit
        (setq tmp5 (cons (cons prop nil) tmp5)))))))
    acc))))))
(seq
 (do ((tmp8 old (cdr tmp8)) (oldPair nil))
    ((or (atom tmp8)
      (progn (setq oldPair (car tmp8)) nil)
      (progn
       (setq name (car oldPair))
       (setq r (cdr oldPair))
       oldPair)
      nil)
   nil)
(seq
  (exit
   (cond
    ((and r (null (lassq name new)))
     (exit
      (setq acc (cons oldPair acc)))))))))
(setq res (nreverse acc))
(Cond
  ((and (boundp '|$reportundo|) $reportundo)
   (reportundo res))
  (exit res)))))))
defun clearFrame

[clearCmdAll p771]
[$frameRecord p35]
[$previousBindings p36]

— defun clearFrame —

(defun |clearFrame| ()
  (declare (special |$frameRecord| |$previousBindings|))
  (|clearCmdAll|)
  (setq |$frameRecord| nil)
  (setq |$previousBindings| nil))
25.62 )what Command

what man page

— what.help —

====================================================================
A.28. )what
====================================================================

User Level Required: interpreter

Command Syntax:

- )what categories pattern1 [pattern2 ...]
- )what commands pattern1 [pattern2 ...]
- )what domains pattern1 [pattern2 ...]
- )what operations pattern1 [pattern2 ...]
- )what packages pattern1 [pattern2 ...]
- )what synonym pattern1 [pattern2 ...]
- )what things pattern1 [pattern2 ...]
- )apropos pattern1 [pattern2 ...]

Command Description:

This command is used to display lists of things in the system. The patterns are all strings and, if present, restrict the contents of the lists. Only those items that contain one or more of the strings as substrings are displayed. For example,

)what synonym

displays all command synonyms,

)what synonym ver

displays all command synonyms containing the substring ‘ver’,

)what synonym ver pr

displays all command synonyms containing the substring ‘ver’ or the substring ‘pr’. Output similar to the following will be displayed

----------------- System Command Synonyms -----------------

user-defined synonyms satisfying patterns:

   ver pr


Several other things can be listed with the )what command:

categories displays a list of category constructors.
commands displays a list of system commands available at your user-level. Your user-level is set via the )set userlevel command. To get a description of a particular command, such as ‘‘)what’’, issue )help what.
domains displays a list of domain constructors.
operations displays a list of operations in the system library.
   It is recommended that you qualify this command with one or more patterns, as there are thousands of operations available. For example, say you are looking for functions that involve computation of eigenvalues. To find their names, try )what operations eig. A rather large list of operations is loaded into the workspace when this command is first issued. This list will be deleted when you clear the workspace via )clear all or )clear completely. It will be re-created if it is needed again.
packages displays a list of package constructors.
synonym lists system command synonyms.
things displays all of the above types for items containing the pattern strings as substrings. The command synonym )apropos is equivalent to )what things.

Also See:
o )display
o )set
o )show

---

 affection $whatOptions

 — initvars —

(defvar |$whatOptions| '(|operations| |categories| |domains| |packages|
   |commands| |synonyms| |things|))

---

Also See:
- (25.18 p 796) “display”
- (25.51 p 1013) “set”
- (25.52 p 1019) “show”
defun what

[whatSpad2Cmd p1091]

---

defun what

(defun |what| (l)
  (|whatSpad2Cmd| l))

---

defun whatSpad2Cmd,fixpat

[qcar p??]
[downcase p1206]

---

defun whatSpad2Cmd,fixpat

(defun |whatSpad2Cmd,fixpat| (x)
  (let (xp)
    (if (and (consp x) (progn (setq xp (qcar x)) t))
      (downcase xp)
      (downcase x))))

---

defun whatSpad2Cmd

[reportWhatOptions p1093]
[selectOptionLC p751]
[sayKeyedMsg p27]
[seq p??]
[exit p??]
[whatSpad2Cmd,fixpat p1091]
[whatSpad2Cmd p1091]
[filterAndFormatConstructors p1096]
[whatCommands p1093]
[apropos p1097]
[printSynonyms p745]
[$e p247]
[$whatOptions p1090]
defun whatSpad2Cmd (arg)
  (prog (|$e| |$key0| key args)
    (declare (special |$e| |$whatOptions|))
    (return
      (seq
        (progn
          (setq |$e| |$EmptyEnvironment|)
          (cond
            ((null arg) (|reportWhatOptions|))
            (t
              (setq |key0| (car arg))
              (setq args (cdr arg))
              (setq key (|selectOptionLC| |key0| |$whatOptions| nil))
              (cond
                ((null key)
                 (|sayKeyedMsg|
                  (format nil
                    "Your argument is not valid for the )what system command. \%1 \%1 ~
                    Use the )show system command to display the operations for a ~
                    constructor. Use the )display operations system command to see ~
                    information about an operation. These may be abbreviated to ~
                    )sh and )d op, respectively.")
                 nil))
                (t
                 (setq args
                   (prog (t0)
                     (setq t0 nil)
                     (return
                      (do ((t1 args (cdr t1)) (p nil))
                        ((or (atom t1) (progn (setq p (car t1)) nil))
                          (nreverse0 t0))
                      (seq
                        (exit
                         (setq t0 (cons (|whatSpad2Cmd,fixpat| p) t0)))))))))
                (seq
                  (cond
                    ((eq key '|things|)
                     (do ((t2 |$whatOptions| (cdr t2)) (opt nil))
                         ((or (atom t2) (progn (setq opt (CAR t2)) nil)) nil)
                      (seq
                       (exit
                        (cond
                          ((null (member opt '|things|))
                           (exit (|whatSpad2Cmd| (cons opt args)))))))))))
                ((eq key '|categories|)
                 (|filterAndFormatConstructors| '|category| "Categories" args))
    )))
(eq key 'commands) (whatCommands args)
((eq key 'domains)
 (filterAndFormatConstructors 'domain "Domains" args))
((eq key 'operations)
 (apropos args))
((eq key 'packages)
 (filterAndFormatConstructors 'package "Packages" args))
(t
 (cond ((eq key 'synonyms)
 (printSynonyms args)))))))))))))))

defun Show keywords for )what command

[sayBrightly p??]
[whatOptions p1090]

— defun reportWhatOptions —

(defun reportWhatOptions ()
 (let (optlist)
 (declare (special whatOptions))
 (setq optlist
 (reduce #'append
 (mapcar #'(lambda (x) `(|%l| " " ,x)) whatOptions)))
 (sayBrightly
 ('" )what" "argument keywords are", optlist
 |%l| " or abbreviations thereof." |%l| |%l| " Issue " )what ?"
 "for more information."))))))

defun The )what commands implementation

[strconc p??]
[specialChar p1126]
[filterListOfStrings p1094]
[commandsForUserLevel p720]
[sayMessage p??]
[blankList p??]
[sayAsManyPerLineAsPossible p??]
[say p??]
[sayKeyedMsg p27]
defun whatCommands (patterns)
  (let (label ell)
    (declare (special \$systemCommands \$linelength \$UserLevel))
    (setq label
      (strconc \"System Commands for User Level: \"
        (princ-to-string \$UserLevel)))
    (format t \("v,,,'-:@< ~a ~>\)" (- \$linelength 2) label)
    (setq ell
      (filterListOfStrings patterns
       (mapcar \#'princ-to-string (commandsForUserLevel \$systemCommands))))
    (when patterns
      (if ell
        (sayMessage
         `(\"System commands at this level matching patterns:" \%l \" \"
          ,0(append (blankList patterns) (list nil))))
        (sayMessage
         `(\"No system commands at this level matching patterns:" \%l \" \"
          ,0(append (blankList patterns) (list nil))))))
    (when ell
      (sayAsManyPerLineAsPossible ell)
      (say " ")
    (unless patterns
      (sayKeyedMsg
       (format nil
        \"For more information about individual commands, use the )help ~
        system command followed by the command name or the command name ~
        followed by a question mark. Some commands (such as )lisp ) may ~
        require the )help lisp format. For example, issue )help help or ~
        )help %x1 ? to find out more about the help command itself.\")
       nil))))

defun Find all names contained in a pattern

Names and patterns are lists of strings. This returns a list of strings in names that contains
any of the strings in the patterns [satisfiesRegularExpressions]

— defun filterListOfStrings —

(defun filterListOfStrings (patterns names)
defun Find function of names contained in pattern

The argument names and patterns are lists of strings. The argument fn is something like CAR or CADR. This returns a list of strings in names that contains any of the strings in patterns.

[satisfiesRegularExpressions p1095]

— defun filterListOfStringsWithFn —

(defun filterListOfStringsWithFn (patterns names fn)
  (let (result)
    (if (or (null patterns) (null names))
      names
      (dolist (name (reverse names) result)
        (when (satisfiesRegularExpressions (funcall fn name) patterns)
          (push name result))))))

———

defun satisfiesRegularExpressions

[strpos p1196]

— defun satisfiesRegularExpressions —

(defun satisfiesRegularExpressions (name patterns)
  (let ((dname (downcase (copy name))))
    (dolist (pattern patterns)
      (when (strpos pattern dname 0 "@")
        (return-from nil t))))))

———
defun filterAndFormatConstructors

| sayMessage p?? |
| blankList p?? |
| pp2Cols p?? |
| specialChar p1126 |
| filterListOfStringsWithFn p1095 |
| whatConstructors p1097 |
| function p?? |
| $linelength p983 |

— defun filterAndFormatConstructors —

(defun |filterAndFormatConstructors| (constrType label patterns)
  (prog (l)
    (declare (special $linelength))
    (return
     (progn
      (format t "\~v,,,'-:@< \~a ~>~\%") (- $linelength 2) label)
      (setq l
       (|filterListOfStringsWithFn| patterns
        (|whatConstructors| constrType)
        #'cdr))
      (cond (patterns
        (cond
          (null l)
          (|sayMessage|
           (cons " No "
            (cons label
             (cons " with names matching patterns:")
             (cons '(%l)
              (cons " 
               (append (|blankList| patterns)
                (cons " " nil))))))))
          (t
           (|sayMessage|
            (cons label
             (cons " with names matching patterns:")
             (cons '(%l)
              (cons " 
               (append (|blankList| patterns)
                (cons " " nil)))))))))
      (cond (l (|pp2Cols| l))))

      ——
defun whatConstructors

[boot-equal p??]
[getdatabase p1156]
[seq p??]
[msort p??]
[exit p??]

— defun whatConstructors —

(defun whatConstructors (constrType)
  (prog nil
    (return
     (seq
      (msort
        (prog (t0)
          (setq t0 nil)
          (return
           (do ((t1 (allConstructors) (cdr t1)) (con nil))
             ((or (atom t1) (progn (setq con (car t1)) nil)) (nreverse0 t0))
               (seq
                (exit
                 (cond
                   ((equal (getdatabase con 'constructorkind) constrType)
                    (setq t0
                      (cons
                        (cons
                          (getdatabase con 'abbreviation)
                          (string con))
                        t0))))))))))))))

Display all operation names containing the fragment

Argument l is a list of operation name fragments. This displays all operation names containing these fragments.

[allOperations p1179]
[filterListOfStrings p1094]
[seq p??]
[exit p??]
[downcase p1206]
[sayMessage p20]
[sayAsManyPerLineAsPossible p??]
[msort p??]
[sayKeyedMsg p27]
— defun apropos —

(defun apropos (arg)
  "Display all operation names containing the fragment"
  (prog (ops)
    (return
     (seq
      (progn
       (setq ops
         (cond
           ((null arg) (allOperations))
           (t
            (filterListOfStrings
             (prog (t0)
               (setq t0 nil)
               (return
                (do ((t1 arg (cdr t1)) (p nil))
                    ((or (atom t1) (progn (setq p (car t1)) nil))
                     (nreverse0 t0))
                 (seq (exit (setq t0 (cons (downcase (princ-to-string p)) t0)))))
               (allOperations))))
             (cond
              (ops
               (sayMessage "Operations whose names satisfy the above pattern(s):")
               (sayAsManyPerLineAsPossible (msort ops))
               (sayKeyedMsg
                (format nil "%l To get more information about an operation such as %1, issue ~
                  the command )display op %1"
                (cons (car ops) nil)))
              (t
               (sayMessage " There are no operations containing those patterns")
               nil)))))
    ))
25.63  )workfiles Command

workfiles man page

defun workfiles

(workfilesSpad2Cmd p1099)

—— defun workfiles ——

(defun workfiles (l)
  (workfilesSpad2Cmd l))

————

defun workfilesSpad2Cmd

(throwKeyedMsg p??)
(selectOptionLC p751)
.pathname p1192)
.delete p??)
(makeInputFilename p1129)
.sayKeyedMsg p27)
.namestring p1190)
.updateSourceFiles p806)
.say p??)
.specialChar p1126)
.sortby p??)
.sayBrightly p??)
.$options p??)
.$sourceFiles p??)
.$linelength p983)

—— defun workfilesSpad2Cmd ——

(defun workfilesSpad2Cmd (args)
  (let (deleteflag type flist type1 fl)
    (declare (special "$options" "$sourceFiles" $linelength))
    (cond
      (args
       (throwKeyedMsg)
       (format nil
                "Arguments are not allowed for the )workfiles system command. ~
                 The )boot, )lisp, )meta and )delete options may be used with this ~
                 command, however. Issue )help workfiles for more information.")))
CHAPTER 25. SYSTEM COMMAND HANDLING

nil))
(t
  (setq deleteflag nil)
  (do ((t0 |$options| (cdr t0)) (t1 nil))
    ((or (atom t0)
         (progn (setq t1 (car t0)) nil)
         (progn (progn (setq type (car t1)) t1) nil))
     nil)
  (setq type1
    (|selectOptionLC| type '(|boot| |lisp| |meta| |delete|) nil))
  (cond
    ((null type1)
      (|throwKeyedMsg|
        (format nil
          "%1 is not an allowable option for the )workfiles system command. ~
          The )boot, )lisp, )meta and )delete options may be used with this ~
          command, however. Issue )help workfiles for more information.")
      (cons type nil)))
    ((eq type1 '|delete|) (setq deleteflag t)))
  (do ((t2 |$options| (cdr t2)) (t3 nil))
    ((or (atom t2)
         (progn (setq t3 (CAR t2)) nil)
         (progn
          (setq type (car t3))
          (setq flist (cdr t3)) t3)
     nil))
    nil)
  (setq type1 (|selectOptionLC| type '(|boot| |lisp| |meta| |delete|) nil))
  (unless (eq type1 '|delete|)
    (dolist (file flist)
      (setq fl (|pathname| (list file type1 "*")))
      (cond
        (deleteflag
          (setq |$sourceFiles| (|delete| fl |$sourceFiles|))
          ((null (makeInputFilename fl))
            (|sayKeyedMsg|
              (format nil
                "The file %1 will not be added to the list of working source ~
                files because the file does not exist.")
                (list (|namestring| fl))))
          (t (|updateSourceFiles| fl))))))))
(say " ")
(format t "\-v,,,'-:@<~a~>~%" (- $linelength 2)
  " User-specified work files ")
(say " ")
(if (null |$sourceFiles|)
  (say " no files specified")
  (progn
    (setq |$sourceFiles| (sortby '|pathnameType| |$sourceFiles|)))
(do ((t5 (cdr t5)) (fl nil))
   ((or (atom t5) (progn (setq fl (car t5)) nil)) nil)
   (|sayBrightly| (list " " (|namestring| f1))))))))


Chapter 26

Handlers for Special Forms

This file contains the functions which do type analysis and evaluation of special functions in the interpreter. Special functions are ones which are not defined in the algebra code, such as assignment, construct, COLLECT and declaration.

Operators which require special handlers all have a LISP “up” property which is the name of the special handler, which is always the word “up” followed by the operator name. If an operator has this “up” property the handler is called automatically from bottomUp instead of general modemap selection.

The up handlers are usually split into two pieces, the first is the up function itself, which performs the type analysis, and an “eval” function, which generates (and executes, if required) the code for the function.

The up functions always take a single argument, which is the entire attributed tree for the operation, and return the modeSet of the node, which is a singleton list containing the type computed for the node.

The eval functions can take any arguments deemed necessary. Actual evaluation is done if $genValue$ is true, otherwise code is generated.

(See the function analyzeMap for other things that may affect what is generated in these functions.)

These functions are required to do two things:

1. do a putValue on the operator vector with the computed value of the node, which is a triple. This is usually done in the eval functions.

2. do a putModeSet on the operator vector with a list of the computed type of the node. This is usually done in the up functions.

There are several special modes used in these functions:

1. Void is the mode that should be used for all statements that do not otherwise return values, such as declarations, loops, IF-THEN’s without ELSE’s, etc..
2. $\texttt{NoValueMode}$ and $\texttt{ThrowAwayMode}$ used to be used in situations where Void is now used, and are being phased out completely.

```lisp
(defun getAndEvalConstructorArgument
  [getValue p??]
  [objMode p448]
  [isWrapped p??]
  [objVal p448]
  [isLocalVar p??]
  [compFailure p??]
  [mkObjWrap p447]
  [timedEVALFUN p??]

  — defun getAndEvalConstructorArgument —

  (defun |getAndEvalConstructorArgument| (tree)
    (let (triple)
      (setq triple (|getValue| tree))
      (cond
        ((eq (|objMode| triple) '(|Domain|)) triple)
        ((|isWrapped| (|objVal| triple)) triple)
        ((|isLocalVar| (|objVal| triple))
         (|compFailure| " Local variable or parameter used in type")
        (t
         (mkObjWrap (|timedEVALFUN| (|objVal| triple) (|objMode| triple)))))))

----------

(defun replaceSharps
  Replaces all sharps in x by the arguments of domain d. Replaces all replaces the triangle variables.

  [subCopy p??]
  [$\texttt{TriangleVariableList}$ p??]
  [$\texttt{FormalMapVariableList}$ p??]

  — defun replaceSharps —

  (defun |replaceSharps| (x d)
    (let (sl)
      (declare (special |$\texttt{TriangleVariableList}$| |$\texttt{FormalMapVariableList}$|))
      (loop for e in (rest d) for var in |$\texttt{FormalMapVariableList}$|
        do (setq sl (cons (cons var e) sl)))
      (setq x (|subCopy| x sl)))
```
(setq sl nil)
(loop for e in (rest d) for var in |$TriangleVariableList|
    do (setq sl (cons (cons var e) sl)))

(defun isDomainValuedVariable

  Returns the value of form if form is a variable with a type value.

  (identp p1197)
  (get p??)
  (member p1198)
  (objMode p448)
  (objValUnwrap p448)
  ($e p247)
  ($env p247)
  ($InteractiveFrame p23)

    — defun isDomainValuedVariable —

(defun |isDomainValuedVariable| (form)
  (let (val)
    (declare (special |$e| |$env| |$InteractiveFrame|))
    (when (and (identp form)
                (setq val
                      (or (|get| form '|value| |$InteractiveFrame|)
                          (and (consp |$env|) (|get| form '|value| |$env|))
                          (and (consp |$e|) (|get| form '|value| |$e|))))
      (|member| (|objMode| val) '((|Domain|) (|SubDomain| (|Domain|))))
      (|objValUnwrap| val))))

(defun evalCategory

  (ofCategory p649)
  (isPartialMode p650)

    — defun evalCategory —

(defun |evalCategory| (d c)
  (or (|isPartialMode| d) (|ofCategory| d c)))
Chapter 27

Handling input files

defun Handle .axiom.input file

[/editfile p781]

— defun readSpadProfileIfThere —

(defun |readSpadProfileIfThere| ()
(let ((file (list '|.axiom| '|input|)))
 (declare (special /editfile))
 (when (makeInputFilename file) (setq /editfile file) (/rq))))

defvar boot-line-stack

— initvars —

(defvar boot-line-stack nil "List of lines returned from preparse")

defvar in-stream

— initvars —

(defvar in-stream t "Current input stream.")
defvar out-stream

— initvars —
(defvar out-stream t "Current output stream.")

defvar file-closed

— initvars —
(defvar file-closed nil "Way to stop EOF tests for console input.")

defvar echo-meta

— initvars —
(defvar echo-meta nil "T if you want a listing of what has been read.")

defvar $noSubsumption

— initvars —
(defvar $noSubsumption t)
defvar $envHashTable

The $envHashTable variable is a hashtable that optimizes lookups in the environment, which normally involve search. This gets populated in the addBinding function.

— initvars —

(defvar $envHashTable nil)

———

defun Dynamically add bindings to the environment

[getProplist p1110]
[addBindingInteractive p1113]
hput p1194]
$InteractiveMode p247]
$envHashTable p1109]

— defun addBinding —

(defun addBinding (var proplist e)
(let (tailContour tailEnv tmp1 curContour lx)
(declare (special $InteractiveMode $envHashTable))
(let (curContour (caar e))
(tailContour (cdar e))
(tailEnv (cdr e))
(cond ((eq proplist (getProplist var e)) e)
(t
  (when $envHashTable
    (do ((prop proplist (cdr prop)) (u nil))
      ((or (atom prop)
          (progn (setq u (car prop)) nil))
       nil)
    (hput $envHashTable (list var (car u)) t)))))
(cond
  ($InteractiveMode (addBindingInteractive var proplist e))
  (t
    (when (and (consp curContour)
      (progn
        (setq tmp1 (qcar curContour))
        (and (consp tmp1) (equal (qcar tmp1) var)))
      (setq curContour (cdr curContour)))
    (setq lx (cons var proplist))
    (cons (cons (cons lx curContour) tailContour) tailEnv))))))

———
defun Fetch a property list for a symbol from CategoryFrame

(defun getProplist (x e)
 (let ((u pl)
          (declare (special "$CategoryFrame"))
          (cond
            ((null (atom x)) (getProplist (car x) e))
            ((setq u (search x e)) u)
            ((setq pl (search x "$CategoryFrame")) pl))))

----------

defun Search for a binding in the environment list

(defun searchCurrentEnv (x currentEnv)
 (prog (u signal)
       (searchCurrentEnv x currentEnv)
       (searchTailEnv x tailEnv))))

----------

defun Search for a binding in the current environment

searchCurrentEnv(x, currentEnv) ==
  for contour in currentEnv repeat
    if u := assq(x, contour) then return (signal := u)
  kdr signal

----------

defun searchCurrentEnv —

(defun searchCurrentEnv (x currentEnv)
 (prog (u signal)
(return
  (seq
    (progn
      (do ((thisenv currentEnv (cdr thisenv)) (contour nil))
          ((or (atom thisenv) (progn (setq contour (car thisenv)) nil)) nil)
        (seq
          (exit
            (cond
              ((setq u (assq x contour)) (return (setq signal u)))
              (t nil))))))
      (ifcdr signal))))

---

defun searchTailEnv

; searchTailEnv(x,e) ==
; for env in e repeat
;  signal:=
;  for contour in env repeat
;    if (u:= ASSQ(x,contour)) and ASSQ("FLUID",u) then return (signal:= u)
;    if signal then return signal
;  KDR signal

[assq p1200]
(cond
  (signal (return signal))
  (t nil)))))
(ifcdr signal))))
Chapter 28

File Parsing

defun Bind a variable in the interactive environment

[assq p1200]

— defun addBindingInteractive —

(defun |addBindingInteractive| (var proplist e)
(let ((curContour (caar e)) u)
(cond
  ((setq u (assq var curContour)) (rplacd u proplist) e)
  (t (rplac (caar e) (cons (cons var proplist) curContour)) e))))

——

defvar line-handler

— initvars —

(defun |initvars|)

(defparameter line-handler 'next-META-line "Who grabs lines for us.")

——

defvar $spad-errors

— initvars —

1113
(defvar $spad_errors (vector 0 0 0))

------

defvar xtokenreader

| initvars |

(defvar xtokenreader 'spadtok)

------

defun Initialize the spad reader

(defun Initialize the spad reader ()
  (declare (special $spad_errors spaderrorstream *standard-output* xtokenreader line-handler meta-error-handler file-closed boot-line-stack))
  (setq $spad_errors (vector 0 0 0))
  (setq spaderrorstream *standard-output*)
  (setq xtokenreader 'get-BOOT-token)
  (setq line-handler 'next-BOOT-line)
  (setq meta-error-handler 'spad-syntax-error)
  (setq file-closed nil)
  (next-lines-clear)
  (ioclear))

------
defun spad-syntax-error

(bumpererrorcount &rest byebye)
"Print syntax error indication, underline character, scrub line."
(declare (special debugmode byebye))
(bumpererrorcount ’|syntax|)
(cond ((and (eq debugmode ’yes) (not(consoleinputp in-stream)))
(spad-long-error))
((spad-short-error)))
(ioclear)
(throw ’spad_reader nil))

— defun spad-syntax-error —

defun spad-long-error

(spad-error-loc &rest byebye)

(defun spad-long-error ()
(declare (special spaderrorstream))
(spad-error-loc spaderrorstream)
(iostat)
(unless (equal out-stream spaderrorstream)
(spad-error-loc out-stream)
(terpri out-stream)))

— defun spad-long-error —
defun spad-short-error

(defun spad-short-error ()
  (if (line-past-end-p Current-Line)
      (format t "The current line is empty."
      (progn
        (format t "The current line is:"~%"
        (line-print current-line))))))

defun spad-error-loc

(defun spad-error-loc (str)
  (format str "******** Boot Syntax Error detected ********"))

defun iostat

(defun iostat ()
  "Tell me what the current state of the parsing world is."
  (declare (special $boot $spad))
  (if (line-past-end-p Current-Line)
      (format t "The current line is empty."

(progn
  (format t "The current line is:" ~%)
  (line-print current-line))
(if (or $boot $spad) (next-lines-show))
token-stack-show
nil)

defun next-lines-show
[boot-line-stack p1107]
  — defun next-lines-show —
(defun next-lines-show ()
  (declare (special boot-line-stack))
  (and boot-line-stack (format t "Currently preparsed lines are:" ~%))
  (mapcar #'(lambda (line)
                (format t "~5D> ~A~%" (car line) (cdr line)))
           boot-line-stack))

defun token-stack-show
[token-type p??]
  [valid-tokens p??]
  [current-token p??]
  [next-token p??]
  [prior-token p??]
  — defun token-stack-show —
(defun token-stack-show ()
  (if (= valid-tokens 0)
      (format t "There are no valid tokens." ~%)
      (format t "The number of valid tokens is ~S." ~% valid-tokens))
  (when (> valid-tokens 0)
    (format t "The current token is~%")
    (describe current-token))
  (when (> valid-tokens 1)
    (format t "The next token is~%")
    (describe next-token))
  (when (token-type prior-token)
(format t "The prior token was\"\%")
(describe prior-token)))

---

defun ioclear

The IO state manipulation routines assume that

- one I/O stream pair is in effect at any moment
- there is a current line
- there is a current token and a next token
- there is a reduction stack

---

defun ioclear

(defun ioclear (&optional (in t) (out t))
  (declare (special current-fragment current-line $boot $spad)
    (ignore in out))
  (setq current-fragment nil)
  (line-clear current-line)
  (token-install nil nil current-token nil)
  (token-install nil nil next-token nil)
  (token-install nil nil prior-token nil)
  (reduce-stack-clear)
  (if (or $boot $spad) (next-lines-clear))
  nil)

---

defun Set boot-line-stack to nil

(defun next-lines-clear)

---

defun next-lines-clear
(defun next-lines-clear ()
  (setq boot-line-stack nil))
Chapter 29

Handling output

29.1 Special Character Tables

defvar $defaultSpecialCharacters

— initvars —

(defvar |$defaultSpecialCharacters| (list
(int-char 28) ; upper left corner
(int-char 27) ; upper right corner
(int-char 30) ; lower left corner
(int-char 31) ; lower right corner
(int-char 79) ; vertical bar
(int-char 45) ; horizontal bar
(int-char 144) ; APL quad
(int-char 173) ; left bracket
(int-char 189) ; right bracket
(int-char 192) ; left brace
(int-char 208) ; right brace
(int-char 59) ; top box tee
(int-char 62) ; bottom box tee
(int-char 63) ; right box tee
(int-char 61) ; left box tee
(int-char 44) ; center box tee
(int-char 224))) ; back slash

———
defvar $plainSpecialCharacters0

— initvars —

(defvar |$plainSpecialCharacters0| (list
(int-char 78) ; upper left corner (+)
(int-char 78) ; upper right corner (+)
(int-char 78) ; lower left corner (+)
(int-char 78) ; lower right corner (+)
(int-char 79) ; vertical bar
(int-char 96) ; horizontal bar (-)
(int-char 111) ; APL quad (?)
(int-char 173) ; left bracket
(int-char 189) ; right bracket
(int-char 192) ; left brace
(int-char 208) ; right brace
(int-char 78) ; top box tee (+)
(int-char 78) ; bottom box tee (+)
(int-char 78) ; right box tee (+)
(int-char 78) ; left box tee (+)
(int-char 224))) ; back slash

— —

defvar $plainSpecialCharacters1

— initvars —

(defvar |$plainSpecialCharacters1| (list
(int-char 107) ; upper left corner (,,)
(int-char 107) ; upper right corner (,,)
(int-char 125) ; lower left corner (‘)
(int-char 125) ; lower right corner (‘)
(int-char 79) ; vertical bar
(int-char 96) ; horizontal bar (-)
(int-char 111) ; APL quad (?)
(int-char 173) ; left bracket
(int-char 189) ; right bracket
(int-char 192) ; left brace
(int-char 208) ; right brace
(int-char 78) ; top box tee (+)
(int-char 78) ; bottom box tee (+)
(int-char 78) ; right box tee (+)
(int-char 78) ; left box tee (+)
defvar $plainSpecialCharacters2

— initvars —

(defvar $plainSpecialCharacters2 (list
  (int-char 79) ; upper left corner (|)
  (int-char 79) ; upper right corner (|)
  (int-char 79) ; lower left corner (|)
  (int-char 79) ; lower right corner (|)
  (int-char 79) ; vertical bar
  (int-char 96) ; horizontal bar (-)
  (int-char 111) ; APL quad (?)
  (int-char 173) ; left bracket
  (int-char 189) ; right bracket
  (int-char 192) ; left brace
  (int-char 208) ; right brace
  (int-char 78) ; top box tee (+)
  (int-char 78) ; bottom box tee (+)
  (int-char 78) ; right box tee (+)
  (int-char 78) ; left box tee (+)
  (int-char 78) ; center box tee (+)
  (int-char 224))) ; back slash

defvar $plainSpecialCharacters3

— initvars —

(defvar $plainSpecialCharacters3 (list
  (int-char 96) ; upper left corner (-)
  (int-char 96) ; upper right corner (-)
  (int-char 96) ; lower left corner (-)
  (int-char 96) ; lower right corner (-)
  (int-char 79) ; vertical bar
  (int-char 96) ; horizontal bar (-)
  (int-char 111) ; APL quad (?)
  (int-char 173) ; left bracket
  (int-char 189) ; right bracket
  (int-char 192) ; left brace
  (int-char 208) ; right brace
  (int-char 78) ; top box tee (+)
  (int-char 78) ; bottom box tee (+)
  (int-char 78) ; right box tee (+)
  (int-char 78) ; left box tee (+)
  (int-char 78) ; center box tee (+)
  (int-char 224))) ; back slash
defvar $plainRTspecialCharacters

— initvars —

(defvar |$plainRTspecialCharacters| (list
QUOTE +) ; upper left corner (+)
QUOTE +) ; upper right corner (+)
QUOTE +) ; lower left corner (+)
QUOTE +) ; lower right corner (+)
QUOTE \|) ; vertical bar
QUOTE -) ; horizontal bar (-)
QUOTE ?) ; APL quad (?)
QUOTE [) ; left bracket
QUOTE ]) ; right bracket
QUOTE {) ; left brace
QUOTE }) ; right brace
QUOTE +) ; top box tee (+)
QUOTE +) ; bottom box tee (+)
QUOTE +) ; right box tee (+)
QUOTE +) ; left box tee (+)
QUOTE +) ; center box tee (+)
QUOTE \\))) ; back slash

defvar $RTspecialCharacters

— initvars —

(defvar |$RTspecialCharacters| (list
(intern (string (code-char 218))) ;-- upper left corner (+)

29.1. SPECIAL CHARACTER TABLES

(intern (string (code-char 191))) ;-- upper right corner (+)
(intern (string (code-char 192))) ;-- lower left corner (+)
(intern (string (code-char 217))) ;-- lower right corner (+)
(intern (string (code-char 179))) ;-- vertical bar
(intern (string (code-char 196))) ;-- horizontal bar (-)
(list (code-char #x1d) (code-char #xe2)) ;-- APL quad (?)

QUOTE []
QUOTE []
QUOTE {}
QUOTE {}

(intern (string (code-char 194))) ;-- top box tee (+)
(intern (string (code-char 193))) ;-- bottom box tee (+)
(intern (string (code-char 180))) ;-- right box tee (+)
(intern (string (code-char 195))) ;-- left box tee (+)
(intern (string (code-char 197))) ;-- center box tee (+)

QUOTE \|

defvar $specialCharacters

– initvars –
(defvar |$specialCharacters| |$RTspecialCharacters|)

– defvar $specialCharacterAlist

– initvars –
(defvar |$specialCharacterAlist|)
'((|ulc| . 0)
  (|urc| . 1)
  (|llc| . 2)
  (|lrc| . 3)
  (|vbar| . 4)
  (|hbar| . 5)
  (|quad| . 6)
  (|lbrk| . 7)
  (|rbrk| . 8)
  (|lbrcl| . 9)
defun Look up a special character code for a symbol

This function looks up a symbol in $specialCharacterAlist, gets the index into the EBCDIC
table, and returns the appropriate character.  TPDHERE: Make this more interna-
tional, not EBCDIC.

(defun specialChar (symbol)
  (let (code)
    (declare (special $specialCharacters $specialCharacterAlist))
    (if (setq code (ifcdr (assq symbol $specialCharacterAlist)))
      (elt $specialCharacters code) "?")))
Chapter 30
Stream and File Handling

defun make-instream

(makeInputFilename p1129)

— defun make-instream —

(defun make-instream (filespec &optional (recnum 0))
  (declare (ignore recnum))
  (cond ((numberp filespec) (make-synonym-stream '*terminal-io*))
    ((null filespec) (error "not handled yet"))
    (t (open (makeInputFilename filespec)
              :direction :input :if-does-not-exist nil))))

—

defun make-outstream

(make-filename p??)

— defun make-outstream —

(defun make-outstream (filespec &optional (width nil) (recnum 0))
  (declare (ignore width) (ignore recnum))
  (cond ((numberp filespec) (make-synonym-stream '*terminal-io*))
    ((null filespec) (error "not handled yet"))
    (t (open (make-filename filespec) :direction :output)))))

—

1127
defun make-appendstream

(make-filename p??)

— defun make-appendstream —

(defun make-appendstream (filespec &optional (width nil) (recnum 0))
"fortran support"
(declare (ignore width) (ignore recnum))
(cond
((numberp filespec) (make-synonym-stream '*terminal-io*))
((null filespec) (error "make-appendstream: not handled yet"))
('else (open (make-filename filespec) :direction :output
:if-exists :append :if-does-not-exist :create))))

—

defun defiostream

— defun defiostream —

(defun defiostream (stream-alist buffer-size char-position)
(declare (ignore buffer-size))
(let ((mode (or (cdr (assoc 'mode stream-alist)) 'input))
(filename (cdr (assoc 'file stream-alist)))
(dev (cdr (assoc 'device stream-alist))))
(if (eq dev 'console) (make-synonym-stream '*terminal-io*)
(let ((strm (case mode
((output o) (open (make-filename filename)
:direction :output))
((input i) (open (makeInputFilename filename)
:direction :input))))
(if (and (numberp char-position) (> char-position 0))
(file-position strm char-position)
strm))))

—

defun shut

[shut is-console (vol9)]

— defun shut —
(defun shut (st)
  (if (is-console st)
      st
    (if (streamp st) (close st) -1)))

------

defun eofp

      — defun eofp —

(defun eofp (stream) (null (peek-char nil stream nil nil)))

------

defun makeStream

[make-appendstream p1128]
[make-outstream p1127]

      — defun makeStream —

(defun |makeStream| (append filename i j)
  (if append
      (make-appendstream filename i j)
    (make-outstream filename i j)))

------

defun Construct a new input file name

      — defun makeInputFilename —

(defun makeInputFilename (filearg &optional (filetype nil))
  (let* ((filename (make-filename filearg filetype))
    (dirname (pathname-directory filename))
    (ft (pathname-type filename))
    (dirs (getDirectoryList ft))
    (newfn nil))
  (if (or (null dirname) (eqcar dirname :relative))
(dolist (dir dirs (probesName filename))
  (when (probes-file (setq newfn (concatenate 'string dir filename)))
    (return newfn)))
  (probesName filename))))

defun getDirectoryList

(defun getDirectoryList (ft &aux (cd (namestring $current-directory)))
  (declare (special $current-directory |$UserLevel| $library-directory-list $directory-list))
  (if (member ft '("nrlib" "daase" "exposed") :test #'string=)
      (if (eq |$UserLevel| '|development|)
          (cons cd $library-directory-list)
        (adjoin cd
         (adjoin (namestring (user-homedir-pathname)) $directory-list
          :test #'string=)
         :test #'string=))))

defun probeName

Sometimes we are given a file and sometimes we are given the name of an Axiom KAF (Keyed-Access File). KAF files are actually directories with a single file called “index.kaf”. We check for the latter case and return the directory name as the filename, per Axiom convention.

(defun probeName (file)
  (when (or (probes-file file)
    (probes-file (concatenate 'string (namestring file) "/index.kaf"))
  (namestring file))))
defun makeFullNamestring

  — defun makeFullNamestring —

(defun makeFullNamestring (filearg &optional (filetype nil))
  (namestring (merge-pathnames (make-filename filearg filetype))))

defun Replace a file by erase and rename

[makeFullNamestring p1131]

  — defun replaceFile —

(defun replaceFile (filespec1 filespec2)
  ($erase (setq filespec1 (makeFullNamestring filespec1)))
  (rename-file (makeFullNamestring filespec2) filespec1))
Chapter 31

The Spad Server Mechanism

defun openserver

This is a cover function for the C code used for communication interface.
— defun openserver —

(defun openserver (name)
  (open_server name))

--------
Chapter 32

Axiom Build-time Functions

defun spad-save

The spad-save function is just a cover function for more lisp system specific save functions. There is no standard name for saving a lisp image so we make one and conditionalize it at compile time.

This function is passed the name of an image that will be saved. The saved image contains all of the loaded functions.

This is used in the src/interp/Makefile.pamphlet in three places:

- creating depsys, an image for compiling axiom.
  
  Some of the Common Lisp code we compile uses macros which are assumed to be available at compile time. The DEPSYS image is created to contain the compile time environment and saved. We pipe compile commands into this environment to compile from Common Lisp to machine dependent code.

  DEPSYS=${OBJ}/${SYS}/bin/depsys

- creating savesys, an image for running axiom.
  
  Once we’ve compile all of the Common Lisp files we fire up a clean lisp image called LOADSYS, load all of the final executable code and save it out as SAVESYS. The SAVESYS image is copied to the ${MNT}/${SYS}/bin subdirectory and becomes the axiom executable image.

  LOADSYS= ${OBJ}/${SYS}/bin/lisp
  SAVESYS= ${OBJ}/${SYS}/bin/interpsys
  AXIOMSYS= ${MNT}/${SYS}/bin/AXIOMsys

- creating debugsys, an image with all interpreted functions loaded.
Occasionally we need to really get into the system internals. The best way to do this is to run almost all of the lisp code interpreted rather than compiled (note that cfuns.lisp and sockio.lisp still need to be loaded in compiled form as they depend on the loader to link with lisp internals). This image is nothing more than a load of the file src/interp/debugsys.lisp.pamphlet. If you need to make test modifications you can add code to that file and it will show up here.

```
DEBUGSYS=${OBJ}/${SYS}/bin/debugsys
[save-system p??]
[$SpadServer p133]
[$openServerIfTrue p131]

--- defun spad-save ---
(defun user::spad-save (save-file)
  (declare (special |$SpadServer| $openServerIfTrue))
  (setq |$SpadServer| nil)
  (setq $openServerIfTrue t)
#+:AKCL
  (system::save-system save-file)
#+:allegro
  (if (fboundp 'boot::restart)
      (excl::dumplisp :name save-file :restart-function #'boot::restart)
      (excl::dumplisp :name save-file))
#+Lucid
  (if (fboundp 'boot::restart)
      (sys::disksave save-file :restart-function #'boot::restart)
      (sys::disksave save-file))
#+:CCL
  (preserve)
)
```
Chapter 33

Exposure Groups

Exposure groups are a way of controlling the namespace available to the user. Certain algebra files are only useful for internal purposes but they contain functions have common names (like “map”). In order to separate the user visible functions from the internal functions the algebra files are collected into “exposure groups”. These large groups are grouped into sets in the variable $globalExposureGroupAlist$.

Exposure group information is kept in the local frame. For more information “The Frame Mechanism” 2.1 on page 3.
Chapter 34

Databases

34.1 Database structure

In order to understand this program you need to understand some details of the structure of the databases it reads. Axiom has 5 databases, the interp.daase, operation.daase, category.daase, and browse.daase.

kaf File Format

This documentation refers to kaf files which are random access files. nrllib files are kaf files (look for nrllib/index.kaf) The format of a random access file is

```
byte-offset-of-key-table
first-entry
second-entry
...
last-entry
((key1 . first-entry-byte-address)
 (key2 . second-entry-byte-address)
 ... 
 (keyN . last-entry-byte-address))
```

The key table is a standard lisp alist.

To open a database you fetch the first number, seek to that location, and (read) which returns the key-data alist. To look up data you index into the key-data alist, find the ith-entry-byte-address, seek to that address, and (read).

For instance, see src/share/algebra/users.daase/index.kaf

One existing optimization is that if the data is a simple thing like a symbol then the nth-entry-byte-address is replaced by immediate data.
Another existing one is a compression algorithm applied to the data so that the very long names don’t take up so much space. We could probably remove the compression algorithm as 64k is no longer considered ’huge’. The database-abbreviation routine handles this on read and write-compress handles this on write.

Indeed, a faster optimization is to simply read the whole database into the image before it is saved. The system would be easier to understand and the interpreter would be faster.

The fastest optimization is to fix the time stamp mechanism which is currently broken. Making this work requires a small bit of coordination at ’make’ time which I forgot to implement.

Database Files

Database files are very similar to kaf files except that there is an optimization (currently broken) which makes the first item a pair of two numbers. The first number in the pair is the offset of the key-value table, the second is a time stamp. If the time stamp in the database matches the time stamp in the image the database is not needed (since the internal hash tables already contain all of the information). When the database is built the time stamp is saved in both the gcl image and the database.

Regarding the ’ancestors field in a category: At database build time there exists a ”ancestors-hash” hash table that gets filled with CATEGORY (not domain) ancestor information. This later provides the information that goes into interp.daase This ”ancestors-hash” does not exist at normal runtime (it can be made by a call to genCategoryTable). Note that the ancestor information in ”ancestors-hash” (and hence interp.daase) involves #1, #2, etc instead of R, Coef, etc. The latter thingies appear in all .nrlib/index.kaf files. So we need to be careful when we )lib categories and update the ancestor info.

This file contains the code to build, open and access the .daase files. This file contains the code to )library nrlibs and asy files

There is a major issue about the data that resides in these databases. the fundamental problem is that the system requires more information to build the databases than it needs to run the interpreter. in particular, modemap.daase is constructed using properties like ”modemaps” but the interpreter will never ask for this information.

So, the design is as follows:

- the modemap.daase needs to be built. this is done by doing a )library on ALL of the nrlib files that are going into the system. this will bring in ”modemap” information and add it to the ”modemaps-hash” hashtable.

- database build proceeds, accessing the ”modemap” property from the hashtables. once this completes this information is never used again.

- the interp.daase database is built. this contains only the information necessary to run the interpreter. note that during the running of the interpreter users can extend the system by do a )library on a new nrlib file. this will cause fields such as ”modemap” to be read and hashed.
Each constructor (e.g. LIST) had one library directory (e.g. LIST.nrlib). This directory contained a random access file called the index.kaf file. These files contain runtime information such as the operationAlist and the ConstructorModemap. At system build time we merge all of these .nrlib/index.kaf files into one database, INTERP.datase. Requests to get information from this database are cached so that multiple references do not cause additional disk i/o.

This database is left open at all times as it is used frequently by the interpreter. One minor complication is that newly compiled files need to override information that exists in this database.

The design calls for constructing a random read (kaf format) file that is accessed by functions that cache their results. When the database is opened the list of constructor-index pairs is hashed by constructor name. A request for information about a constructor causes the information to replace the index in the hash table. Since the index is a number and the data is a non-numeric sexpr there is no source of confusion about when the data needs to be read.

The format of this new database is as follows:

- The list is read at open time and hashed by the car of each item.

The system has been changed to use the property list of the symbols rather than hash tables. Since we already hashed once to get the symbol we need only an offset to get the property list. This also has the advantage that eq hash tables no longer need to be moved during garbage collection.

There are 3 potential speedups that could be done.

- The best would be to use the value cell of the symbol rather than the property list but I'm unable to determine all uses of the value cell at the present time.

- A second speedup is to guarantee that the property list is a single item, namely the database structure. This removes an assoc but leaves one open to breaking the system if someone adds something to the property list. This was not done because of the danger mentioned.

- A third speedup is to make the getdatabase call go away, either by making it a macro or eliding it entirely. This was not done because we want to keep the flexibility of changing the database forms.
The new design does not use hash tables. the database structure contains an entry for each item that used to be in a hash table. initially the structure contains file-position pointers and these are replaced by real data when they are first looked up. the database structure is kept on the property list of the constructor, thus, (get '|DenavitHartenbergMatrix— 'database) will return the database structure object.

Each operation has a property on its symbol name called 'operation which is a list of all of the signatures of operations with that name.

\textbf{defstruct $\$database}$

\begin{verbatim}
(\textbf{defstruct database}
  abbreviation ; interp.
  ancestors ; interp.
  constructor ; interp.
  constructorcategory ; interp.
  constructorkind ; interp.
  constructormodemap ; interp.
  cosig ; interp.
  defaultdomain ; interp.
  modmaps ; interp.
  niladic ; interp.
  object ; interp.
  operationalist ; interp.
  documentation ; browse.
  constructorform ; browse.
  attributes ; browse.
  predicates ; browse.
  sourcefile ; browse.
  parents ; browse.
  users ; browse.
  dependents ; browse.
  spare ; superstition
); database structure
\end{verbatim}

\textbf{defvar *defaultdomain-list*}$

There are only a small number of domains that have default domains. rather than keep this slot in every domain we maintain a list here.

\begin{verbatim}
(\textbf{defvar *defaultdomain-list*} '(\textbf{defvar *defaultdomain-list*} ')
\end{verbatim}
defvar *operation-hash*

— initvars —

(defvar *operation-hash* nil "given an operation name, what are its modmaps?")

defvar *hasCategory-hash*

This hash table is used to answer the question “does domain x have category y?”. This is answered by constructing a pair of (x . y) and doing an equal hash into this table.

— initvars —

(defvar *hasCategory-hash* nil "answers x has y category questions")

defvar *miss*

This variable is used for debugging. If a hash table lookup fails and this variable is non-nil then a message is printed.

— initvars —
(defvar *miss* nil "print out cache misses on getdatabase calls")

Note that constructor category information need only be kept for items of type category. this
will be fixed in the next iteration when the need for the various caches are reviewed.

Note that the *modemaps-hash* information does not need to be kept for system files. these
are precomputed and kept in modemap.data however, for user-defined files these are needed.
Currently these are added to the database for 2 reasons; there is a still-unresolved issue of
user database extensions and this information is used during database build time.

Database streams

This are the streams for the databases. They are always open. There is an optimization
for speeding up system startup. If the database is opened and the ..stream-stamp* variable
matches the position information in the database then the database is NOT read in and is
assumed to match the in-core version.

    defvar *interp-stream*

    — initvars —

    (defvar *interp-stream* nil "an open stream to the interpreter database")

    ——

    defvar *interp-stream-stamp*

    — initvars —

    (defvar *interp-stream-stamp* 0 "*interp-stream* (position . time)")

    ——

    defvar *operation-stream*

    This is indexed by operation, not constructor

    — initvars —
34.1. DATABASE STRUCTURE

(defvar *operation-stream* nil "the stream to operation.daase")

--------

defvar *operation-stream-stamp*

— initvars —

(defvar *operation-stream-stamp* 0 "*operation-stream* (position . time)"

--------

defvar *browse-stream*

— initvars —

(defvar *browse-stream* nil "an open stream to the browser database"

--------

defvar *browse-stream-stamp*

— initvars —

(defvar *browse-stream-stamp* 0 "*browse-stream* (position . time)"

--------

defvar *category-stream*

This is indexed by (domain . category)

— initvars —

(defvar *category-stream* nil "an open stream to the category table"

--------
defvar *category-stream-stamp*

— initvars —

(defvar *category-stream-stamp* 0 "*category-stream* (position . time)"

_____

defvar *allconstructors*

— initvars —

(defvar *allconstructors* nil "a list of all the constructors in the system"

_____

defvar *allOperations*

— initvars —

(defvar *allOperations* nil "a list of all the operations in the system"

_____

defun Reset all hash tables before saving system

[interpopen p1149]
[operationopen p1153]
[browseopen p1151]
[categoryopen p1152]
[initial-getdatabase p1147]
[*sourcefiles* p1164]
[*interp-stream* p1144]
[*operation-stream* p1144]
[*category-stream* p1145]
[*browse-stream* p1145]
[*category-stream-stamp* p1146]
[*operation-stream-stamp* p1145]
[*interp-stream-stamp* p1144]
defun resethashables ()
"set all -hash* to clean values. used to clean up core before saving system"

(defun resethashables ()
  "set all -hash* to clean values. used to clean up core before saving system"
  (declare (special *sourcefiles* *interp-stream* *operation-stream*
      *category-stream* *browse-stream* *category-stream-stamp*
      *operation-stream-stamp* *interp-stream-stamp*
      *allconstructors* *operation-hash* *hascategory-hash*))

  (setq *hascategory-hash* (make-hash-table :test #'equal))
  (setq *operation-hash* (make-hash-table))
  (setq *allconstructors* nil)
  (setq *sourcefiles* nil)
  (setq *interp-stream-stamp* '(0 . 0))
  (interopen)
  (setq *operation-stream-stamp* '(0 . 0))
  (operationopen)
  (setq *browse-stream-stamp* '(0 . 0))
  (browseopen)
  (setq *category-stream-stamp* '(0 . 0))
  (categoryopen);
  (initial-getdatabase)
  (close *interp-stream*)
  (close *operation-stream*)
  (close *category-stream*)
  (close *browse-stream*)
  (gbc t))

----------

defun Preload algebra into saved system

[getdatabase p1156]
[getenviron p255]

--- defun initial-getdatabase ---

(defun initial-getdatabase ()
"fetch data we want in the saved system"
(declare (special hascategory constructormodemapAndoperationalist operation constr)

  (format t "Initial getdatabase\"%
  (setq hascategory '(\
     (\|Equation\| . \|Ring\|)
     (\|Expression\| . \|CoercibleTo\|) (\|Expression\| . \|CommutativeRing\|)
...
CHAPTER 34. DATABASES

(|Expression| . |IntegralDomain|) (|Expression| . |Ring|)
(|Float| . |RetractableTo|)
(|Fraction| . |Algebra|) (|Fraction| . |CoercibleTo|)
(|Fraction| . |OrderedSet|) (|Fraction| . |RetractableTo|)
(|Integer| . |Algebra|) (|Integer| . |CoercibleTo|)
(|Integer| . |ConvertibleTo|) (|Integer| . |LinearlyExplicitRingOver|)
(|Integer| . |RetractableTo|)
(|List| . |CoercibleTo|) (|List| . |FiniteLinearAggregate|)
(|List| . |OrderedSet|)
(|Polynomial| . |CoercibleTo|) (|Polynomial| . |CommutativeRing|)
(|Polynomial| . |ConvertibleTo|) (|Polynomial| . |OrderedSet|)
(|Polynomial| . |RetractableTo|)
(|Symbol| . |CoercibleTo|) (|Symbol| . |ConvertibleTo|)
(|Variable| . |CoercibleTo|)

(dolist (pair hascategory) (getdatabase pair 'hascategory))

(setq constructormodemapAndoperationalist '(

(dolist (con constructormodemapAndoperationalist)
  (getdatabase con 'constructormodemap)
  (getdatabase con 'operationalist))

(setq operation '(
|+| |-| |*| |/| |**| |coerce| |convert| |elt| |equation| |float| |sin| |cos| |map| |SEGMENT|))

(dolist (op operation) (getdatabase op 'operation))

(setq constr '( ;these are sorted least-to-most freq. delete early ones first
|Factored| |SparseUnivariatePolynomialFunctions2| |TableAggregate&|
|RetractableTo&| |RecursiveAggregate&| |UserDefinedPartialOrdering|
|None| |UnivariatePolynomialCategoryFunctions2| |IntegerPrimesPackage|
|SetCategory&| |IndexedExponents| |QuotientFieldCategory&| |Polynomial|
|EltableAggregate&| |PartialDifferentialRing&| |Set|
|UnivariatePolynomialCategory&| |FlexibleArray|
|SparseMultivariatePolynomial| |PolynomialCategory|))
<table>
<thead>
<tr>
<th>DifferentialExtension&amp;</th>
<th>IndexedFlexibleArray</th>
<th>AbelianMonoidRing&amp;</th>
</tr>
</thead>
<tbody>
<tr>
<td>FiniteAbelianMonoidRing&amp;</td>
<td>DivisionRing&amp;</td>
<td>FullyLinearlyExplicitRingOver&amp;</td>
</tr>
<tr>
<td>IndexedVector</td>
<td>IndexedOneDimensionalArray</td>
<td>LocalAlgebra</td>
</tr>
<tr>
<td>Boolean</td>
<td>Field&amp;</td>
<td>Vector</td>
</tr>
<tr>
<td>PolynomialRing</td>
<td>FreeModule</td>
<td>IndexedDirectProductAbelianGroup</td>
</tr>
<tr>
<td>IndexedDirectProductAbelianMonoid</td>
<td>SingletonAsOrderedSet</td>
<td></td>
</tr>
<tr>
<td>SparseUnivariatePolynomial</td>
<td>Fraction</td>
<td>Collection&amp;</td>
</tr>
<tr>
<td>RepeatedSquaring</td>
<td>IntegerNumberSystem&amp;</td>
<td>AbelianSemigroup&amp;</td>
</tr>
<tr>
<td>AssociationList</td>
<td>OrderedRing&amp;</td>
<td>SemiGroup&amp;</td>
</tr>
<tr>
<td>UniqueFactorizationDomain&amp;</td>
<td>EuclideanDomain&amp;</td>
<td>IndexedAggregate&amp;</td>
</tr>
<tr>
<td>GcdDomain&amp;</td>
<td>IntegralDomain&amp;</td>
<td>DifferentialRing&amp;</td>
</tr>
<tr>
<td>UnaryRecursiveAggregate&amp;</td>
<td>OrderedSet&amp;</td>
<td>AbelianGroup&amp;</td>
</tr>
<tr>
<td>Module&amp;</td>
<td>Ring&amp;</td>
<td>StringAggregate&amp;</td>
</tr>
<tr>
<td>ExtensibleLinearAggregate&amp;</td>
<td>PositiveInteger</td>
<td>StreamAggregate&amp;</td>
</tr>
<tr>
<td>IndexedString</td>
<td>IndexedList</td>
<td>ListAggregate&amp;</td>
</tr>
<tr>
<td>Character</td>
<td>String</td>
<td>NonNegativeInteger</td>
</tr>
<tr>
<td>OneDimensionalArrayAggregate&amp;</td>
<td>FiniteLinearAggregate&amp;</td>
<td>PrimitiveArray</td>
</tr>
<tr>
<td>Integer</td>
<td>List</td>
<td>OutputForm</td>
</tr>
</tbody>
</table>

(dolist (con constr)
 (let ((c (concatenate 'string
 (getenviron "AXIOM") "/algebra/
 (string (getdatabase con 'abbreviation)) ".o")))
 (format t " preloading ~a.." c)
 (if (probe-file c)
 (progn
 (put con 'loaded c)
 (load c)
 (format t "loaded.ESTAMP")
 (format t "skipped.ESTAMP")
 (format t ""))
 (format t ""))

### defun Open the interp database

Format of an entry in interp.daase:

(constructor-name
 operationalist
 constructormodemap
 modemaps -- this should not be needed. eliminate it.
 object -- the name of the object file to load for this con.
 constructorcategory -- note that this info is the cadar of the
 constructormodemap for domains and packages so it is stored
 as NIL for them. it is valid for categories.
 niladic -- t or nil directly
 unused
 cosig -- kept directly
constructorkind -- kept directly
defaultdomain -- a short list, for %i
ancestors -- used to compute new category updates

[make-database p??]
[DaaseName p1169]
[$spadroot p133]
[*allconstructors* p1146]
[*interp-stream* p1144]
[*interp-stream-stamp* p1144]

— defun interpopen —

(defun interpopen ()
  "open the interpreter database and hash the keys"
  (declare (special $spadroot *allconstructors* *interp-stream*
    *interp-stream-stamp*))
  (let (constructors pos stamp dbstruct)
    (setq *interp-stream* (open (DaaseName "interp.daase" nil)))
    (setq stamp (read *interp-stream*))
    (unless (equal stamp *interp-stream-stamp*)
      (format t " Re-reading interp.daase")
      (setq *interp-stream-stamp* stamp)
      (setq pos (car stamp))
      (file-position *interp-stream* pos)
      (setq constructors (read *interp-stream*))
    (dolist (item constructors)
      (setq *allconstructors* (adjoin (first item) *allconstructors*))
      (setq dbstruct (make-database))
      (setf (get (car item) 'database) dbstruct)
      (setf (database-operationalist dbstruct) (second item))
      (setf (database-constructormodemap dbstruct) (third item))
      (setf (database-modemaps dbstruct) (fourth item))
      (setf (database-object dbstruct) (fifth item))
      (setf (database-constructorcategory dbstruct) (sixth item))
      (setf (database-niladic dbstruct) (seventh item))
      (setf (database-abbreviation dbstruct) (eighth item))
      (setf (get (eighth item) 'abbreviationfor) (first item)) ;invert
      (setf (database-cosig dbstruct) (ninth item))
      (setf (database-constructorkind dbstruct) (tenth item))
      (setf (database-ancestors dbstruct) (nth 11 item)))
    (format t "&")
  )

This is an initialization function for the constructor database it sets up 2 hash tables, opens
the database and hashes the index values.
There is a slight asymmetry in this code. The sourcefile information for system files is only the filename and extension. For user files it contains the full pathname. When the database is first opened the sourcefile slot contains system names. The lookup function has to prefix the "$spadroot" information if the directory-namestring is null (we don’t know the real root at database build time).

An object-hash table is set up to look up nrlib and ao information. This slot is empty until a user does a )library call. We remember the location of the nrlib or ao file for the users local library at that time. A NIL result from this probe means that the library is in the system-specified place. When we get into multiple library locations this will also contain system files.

**defun Open the browse database**

Format of an entry in browse.daase:

```lisp
(constructorname
 sourcefile
 constructorform
 documentation
 attributes
 predicates
)
```

[$spadroot p133]
[*allconstructors* p1146]
[*browse-stream* p1145]
[*browse-stream-stamp* p1145]

— defun browseopen —

```lisp
(defun browseopen ()
 "open the constructor database and hash the keys"
 (declare (special $spadroot *allconstructors* *browse-stream*
 *browse-stream-stamp*))
 (let (constructors pos stamp dbstruct)
   (setq *browse-stream* (open (DaaseName "browse.daase" nil)))
   (setq stamp (read *browse-stream*))
   (unless (equal stamp *browse-stream-stamp*)
     (format t " Re-reading browse.daase")
     (setq *browse-stream-stamp* stamp)
     (setq pos (car stamp))
     (file-position *browse-stream* pos)
     (setq constructors (read *browse-stream*))
   (dolist (item constructors)
     (unless (setq dbstruct (get (car item) 'database))
       (format t "browseopen:~%")
       (format t "the browse database contains a constructor "~a" item))
```
(format t "that is not in the interp.daase file. we cannot~")
(format t "get the database structure for this constructor and"\n)
(warn "will create a new one")
(setq (get (car item) 'database) (setq dbstruct (make-database)))
(setq *allconstructors* (adjoin item *allconstructors*)))
(setq (database-sourcefile dbstruct) (second item))
(setq (database-constructorform dbstruct) (third item))
(setq (database-documentation dbstruct) (fourth item))
(setq (database-attributes dbstruct) (fifth item))
(setq (database-predicates dbstruct) (sixth item))
(setq (database-parents dbstruct) (seventh item))))
(format t ""\n))

defun Open the category database

[$spadroot$ p133]
[*hasCategory-hash* p1143]
[*category-stream* p1145]
[*category-stream-stamp* p1146]

— defun categoryopen —

(defun categoryopen ()
  "open category.daase and hash the keys"
  (declare (special $spadroot *hasCategory-hash* *category-stream*  
                   *category-stream-stamp*))
  (let (pos keys stamp)
    (setq *category-stream* (open (DaaseName "category.daase" nil)))
    (setq stamp (read *category-stream*))
    (unless (equal stamp *category-stream-stamp*)
      (format t " Re-reading category.daase")
      (setq *category-stream-stamp* stamp)
      (setq pos (car stamp))
      (file-position *category-stream* pos)
      (setq keys (read *category-stream*))
      (setq *hasCategory-hash* (make-hash-table :test #'equal))
      (dolist (item keys)
        (setf (gethash (first item) *hasCategory-hash*) (second item))))
    (format t ""\n))

——
defun Open the operations database

(defun operationopen ()
  "read operation database and hash the keys"
  (declare (special $spadroot *operation-hash* *operation-stream*
     *operation-stream-stamp*))
  (let (operations pos stamp)
    (setq *operation-stream* (open (DaaseName "operation.daase" nil)))
    (setq stamp (read *operation-stream*))
    (unless (equal stamp *operation-stream-stamp*)
      (format t " Re-reading operation.daase")
      (setq *operation-stream-stamp* stamp)
      (setq pos (car stamp))
      (file-position *operation-stream* pos)
      (setq operations (read *operation-stream*))
      (dolist (item operations)
        (setf (gethash (car item) *operation-hash*) (cdr item))))
    (format t "&")))

----

defun Add operations from newly compiled code

(defun addoperations (constructor oldmaps)
  "add ops from a )library domain to *operation-hash*"
  (declare (special *operation-hash*))
  (dolist (map oldmaps) ; out with the old
    (let (oldop op)
      (setq op (car map))
      (setq oldop (getdatabase op 'operation))
      (setq oldop (lisp::delete (cdr map) oldop :test #'equal))
      (setf (gethash op *operation-hash* oldop)))
    (dolist (map (getdatabase constructor 'modemaps)) ; in with the new
      (let (op newmap)
        (setq op (car map))
        (setq op (car map))))
(setq newmap (getdatabase op 'operation))
(setf (gethash op *operation-hash*) (cons (cdr map) newmap))))

---

defun Show all database attributes of a constructor

[getdatabase p1156]

--- defun showdatabase ---

(defun showdatabase (constructor)
  (format t ""%a: ~a%" 'constructorkind
          (getdatabase constructor 'constructorkind))
  (format t ""%a: ~a%" 'cosig
          (getdatabase constructor 'cosig))
  (format t ""%a: ~a%" 'operation
          (getdatabase constructor 'operation))
  (format t ""%a: ~a%" 'constructormodemap
          (pprint (getdatabase constructor 'constructormodemap))
  (format t ""%a: ~a%" 'constructorcategory
          (pprint (getdatabase constructor 'constructorcategory))
  (format t ""%a: ~a%" 'operationalist
          (pprint (getdatabase constructor 'operationalist))
  (format t ""%a: ~a%" 'constructormodemap
          (pprint (getdatabase constructor 'constructormodemap))
  (format t ""%a: ~a%" 'hascategory
          (getdatabase constructor 'hascategory))
  (format t ""%a: ~a%" 'object
          (getdatabase constructor 'object))
  (format t ""%a: ~a%" 'niladic
          (getdatabase constructor 'niladic))
  (format t ""%a: ~a%" 'abbreviation
          (getdatabase constructor 'abbreviation))
  (format t ""%a: ~a%" 'constructor?
          (getdatabase constructor 'constructor?))
  (format t ""%a: ~a%" 'constructor
          (getdatabase constructor 'constructor))
  (format t ""%a: ~a%" 'defaultdomain
          (getdatabase constructor 'defaultdomain))
  (format t ""%a: ~a%" 'ancestors
          (getdatabase constructor 'ancestors))
  (format t ""%a: ~a%" 'sourcefile
          (getdatabase constructor 'sourcefile))
  (format t ""%a: ~a%" 'constructorform
          (getdatabase constructor 'constructorform))
  (format t ""%a: ~a%" 'constructorargs
          (getdatabase constructor 'constructorargs))

---
34.1. DATABASE STRUCTURE

(defun Set a value for a constructor key in the database

[make-database p??]

---

defun setdatabase ---

(defun setdatabase (constructor key value)
  (let (struct)
    (when (symbolp constructor)
      (unless (setq struct (get constructor 'database))
        (setq struct (make-database))
        (setf (get constructor 'database) struct))
      (case key
        (abbreviation
          (setf (database-abbreviation struct) value)
          (when (symbolp value)
            (setf (get value 'abbreviationfor) constructor)))
        (constructorkind
          (setf (database-constructorkind struct) value))))))

---

defun Delete a value for a constructor key in the database

---

defun deldatabase ---

(defun deldatabase (constructor key)
  (when (symbolp constructor)
    (case key
      (abbreviation
        (setf (get constructor 'abbreviationfor nil))))))
defun Get constructor information for a database key

[warn p??]
[$spadroot p133]
[*miss* p1143]
[*hascategory-hash* p??]
[*operation-hash* p1143]
[*browse-stream* p1145]
[*defaultdomain-list* p1142]
[*interp-stream* p1144]
[*category-stream* p1145]
[*hasCategory-hash* p1143]
[*operation-stream* p1144]

— defun getdatabase —

(defun getdatabase (constructor key)
  (declare (special $spadroot) (special *miss*))
  (when (eq *miss* t) (format t "getdatabase call: ~20a ~a~%" constructor key))
  (let (data table stream ignore struct)
    (declare (ignore ignore)
      (special *hascategory-hash* *operation-hash*
        *browse-stream* *defaultdomain-list* *interp-stream*
        *category-stream* *hasCategory-hash* *operation-stream*))
    (when (or (symbolp constructor)
              (and (eq key 'hascategory) (consp constructor)))
      (case key
        ; note that abbreviation, constructorkind and cosig are heavy hitters
        ; thus they occur first in the list of things to check
        (abbreviation
          (setq stream *interp-stream*)
          (when (setq struct (get constructor 'database))
            (setq data (database-abbreviation struct))))
        (constructorkind
          (setq stream *interp-stream*)
          (when (setq struct (get constructor 'database))
            (setq data (database-constructorkind struct))))
        (cosig
          (setq stream *interp-stream*)
          (when (setq struct (get constructor 'database))
            (setq data (database-cosig struct))))
        (operation
          (setq stream *operation-stream*)
          (setq data (gethash constructor *operation-hash*)))
        (constructormodemap
          (setq stream *interp-stream*)
          (when (setq struct (get constructor 'database))
            (setq data (database-constructormodemap struct))))
        (constructorcategory
          (setq stream *interp-stream*)
          (when (setq struct (get constructor 'database))
            (setq data (database-constructorcategory struct))))
    )))
34.1. DATABASE STRUCTURE

(setq stream *interp-stream*)
(when (setq struct (get constructor 'database))
 (setq data (database-constructorcategorystruct))
 (when (null data) ;domain or package then subfield of constructormodemap
  (setq data (cadar (getdatabase constructor 'constructormodemap)))))

(operationalist
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-operationalist struct)))

(modemaps
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-modemaps struct)))

(hascategory
 (setq table *hasCategory-hash*)
 (setq stream *category-stream*)
 (setq data (gethash constructor table)))

(object
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-object struct)))

(niladic
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-niladic struct)))

(constructor?
 (when (setq struct (get constructor 'database))
  (setq data (when (database-operationalist struct) t))))

(superdomain ; only 2 superdomains in the world
 (case constructor
  (|NonNegativeInteger|
   (setq data '([[Integer]] (IF (<= |#1| 0) |false| |true|))))
  (|PositiveInteger|
   (setq data '([[NonNegativeInteger]] (<= 0 |#1)))))))

(constructor
 (when (setq data (get constructor 'abbreviationfor)))
 (defaultdomain
  (setq data (cadr (assoc constructor *defaultdomain-list*)))))

(ancestors
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-ancestors struct)))

(sourcefile
 (setq stream *browse-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-sourcefile struct))))

(constructorform
 (setq stream *browse-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-constructortype struct))))
(constructorargs
  (setq data (cdr (getdatabase constructor 'constructorform))))
(attributes
  (setq stream *browse-stream*)
  (when (setq struct (get constructor 'database))
    (setq data (database-attributes struct))))
(predicates
  (setq stream *browse-stream*)
  (when (setq struct (get constructor 'database))
    (setq data (database-predicates struct))))
/documentation
  (setq stream *browse-stream*)
  (when (setq struct (get constructor 'database))
    (setq data (database-documentation struct))))
(parents
  (setq stream *browse-stream*)
  (when (setq struct (get constructor 'database))
    (setq data (database-parents struct))))
(users
  (setq stream *browse-stream*)
  (when (setq struct (get constructor 'database))
    (setq data (database-users struct))))
(dependents
  (setq stream *browse-stream*)
  (when (setq struct (get constructor 'database))
    (setq data (database-dependents struct))))
(otherwise (warn "~%(GETDATABASE ~a ~a) failed~%" constructor key)))

  (when (numberp data) ; fetch the real data
    (when *miss* (format t "getdatabase miss: ~20a ~a~%" constructor key))
    (file-position stream data)
    (setq data (read stream))
  (case key ; cache the result of the database read
    (operation (setf (gethash constructor *operation-hash*) data))
    (hascategory (setf (gethash constructor *hascategory-hash*) data))
    (constructorkind (setf (database-constructorkind struct) data))
    (cosig (setf (database-cosig struct) data))
    (constructormodemap (setf (database-constructormodemap struct) data))
    (constructorcategory (setf (database-constructorcategory struct) data))
    (operationalist (setf (database-operationalist struct) data))
    (modemaps (setf (database-modemaps struct) data))
    (object (setf (database-object struct) data))
    (niladic (setf (database-niladic struct) data))
    (abbreviation (setf (database-abbreviation struct) data))
    (constructor (setf (database-constructor struct) data))
    (ancestors (setf (database-ancestors struct) data))
    (constructorform (setf (database-constructormform struct) data))
    (attributes (setf (database-attributes struct) data))
    (predicates (setf (database-predicates struct) data))
    (documentation (setf (database-documentation struct) data))
    (parents (setf (database-parents struct) data))
(users (setf (database-users struct) data))
(dependents (setf (database-dependents struct) data))
(sourcefile (setf (database-sourcefile struct) data)))

(case key ; fixup the special cases
  (sourcefile
    (when (and data (string= (directory-namestring data) "")
      (string= (pathname-type data) "spad"))
      (setq data
        (concatenate 'string $spadroot "/../..//src/algebra/" data)))
    (object ; fix up system object pathname
      (if (consp data)
        (setq data
          (if (string= (directory-namestring (car data)) "")
            (concatenate 'string $spadroot "//algebra/" (car data) ".o")
            (car data)))
        (when (and data (string= (directory-namestring data) "")
          (setq data (concatenate 'string $spadroot "//algebra/" data ".o"))))))))

|---|

(defun library top level command

[localdatabase p1160]
[extendLocalLibdb p??]
[serverReadLine is-console (vol9)]
[tersyscommand p724]
[$newConlist p??]
[$options p??]

--- defun library ---

(defun library (args)
  (let (original-directory)
    (declare (special $options |$newConlist|))
    (setq original-directory (get-current-directory))
    (setq |$newConlist| nil)
    (localdatabase args |$options|)
    (clear $newConlist)
    (system::chdir original-directory)
    (tersyscommand)))

---
defun Read a local filename and update the hash tables

The localdatabase function tries to find files in the order of: nrlib/index.kaf [say KeyedMsg p27]

[localnrlib p161]
[$forceDatabaseUpdate p??]
[$ConstructorCache p??]
[*index-filename* p??]

--- defun localdatabase ---

(defun localdatabase (filelist options &optional (make-database? nil))
  "read a local filename and update the hash tables"
  (labels ((processOptions (options)
             (let (only dir noexpose)
               (when (setq only (assoc '|only| options))
                 (setq options (lisp::delete only options :test #'equal))
                 (setq only (cdr only)))
               (when (setq dir (assoc '|dir| options))
                 (setq options (lisp::delete dir options :test #'equal))
                 (setq dir (second dir)))
               (when (null dir)
                 (|sayKeyedMsg|
                   "Ignoring )dir because an explicit directory was not given after )dir." nil))
               (when (setq noexpose (assoc '|noexpose| options))
                 (setq options (lisp::delete noexpose options :test #'equal))
                 (setq noexpose 't))
               (when options
                 (format t " Ignoring unknown )library option: ~a~%" options))
               (values only dir noexpose)))
 (processDir (dirarg thisdir)
               (let (allfiles)
                 (declare (special vmlisp::*index-filename*))
                 (system:chdir (string dirarg))
                 (setq allfiles (directory "*"))
                 (system:chdir thisdir)
                 (mapcan #'(lambda (f)
                               (when (string-equal (pathname-type f) "nrlib")
                                 (list (concatenate 'string (namestring f) "/"
                                      vmlisp::*index-filename*)))))
               (let (thisdir nrlibs object only dir key (|$forceDatabaseUpdate| t) noexpose)
                 (declare (special|$forceDatabaseUpdate| vmlisp::*index-filename*|$ConstructorCache|))
                 (setq thisdir (namestring (truename ".")))
                 (setq noexpose nil)
                 (multiple-value-setq (only dir noexpose) (processOptions options))
                 ; don’t force exposure during database build)
defun Update the database from an nrlib index.kaf file

(defun localnrlib (key nrlib object make-database? noexpose)
  "given a string pathname of an index.kaf and the object update the database"
  (labels (fetchdata (alist in index)
      (let (pos)
        (setq pos (third (assoc index alist :test #'string=)))))
    (if make-database? (setq noexpose t))
    (when dir (setq nrlibs (processDir dir thisdir)))
    (dolist (file filelist)
      (let ((filename (pathname-name file))
            (namedir (directory-namestring file)))
        (unless namedir (setq thisdir (concatenate 'string thisdir "/")))
        (cond
          ((setq file (probe-file (concatenate 'string namedir filename ".nrlib/" vmlisp::*index-filename*))
            (push (namestring file) nrlibs))
            ('else (format t " )library cannot find the file ~a.~%" filename))))
      (dolist (file (nreverse nrlibs))
        (setq key (pathname-name (first (last (pathname-directory file))))
            object (concatenate 'string (directory-namestring file) "code")
            (localnrlib key file object make-database? noexpose))
        (clrhash |$ConstructorCache|)))

defun localnrlib | defun Update the database from an nrlib index.kaf file |

[getdatabase p1156]
[make-database p?]
[addoperations p1153]
[sublis p?]
[updateDatabase p1163]
[installConstructor p?]
[updateCategoryTable p?]
[categoryForm? p?]
[setExposeAddConstr p92]
[startTimingProcess p?]
[loadLibNoUpdate p1183]
[sayKeyedMsg p27]
[$FormalMapVariableList p?]
[*allOperations* p1146]
[*allconstructors* p1146]

|---|---|
(when pos
  (file-position in pos)
  (read in)))))
(let (alist kind (systemdir? nil) pos constructorform oldmaps abbrev dbstruct)
  (declare (special *allOperations* *allconstructors*
               |$FormalMapVariableList|))
  (with-open-file (in nrlib)
    (file-position in (read in))
    (setqalist (read in))
    (setq pos (third (assoc "constructorForm" alist :test #'string=)))
    (file-position in pos)
    (setq constructorform (read in))
    (setq key (car constructorform))
    (setq oldmaps (getdatabase key 'modemaps))
    (setq dbstruct (make-database))
    (setq *allconstructors* (adjoin key *allconstructors*))
    (setq (get key 'database) dbstruct) ; store the struct, side-effect it...
    (setq (database-constructorform dbstruct) constructorform)
    (setq *all1Operations* nil) ; force this to recompute
    (setq (database-object dbstruct) object)
    (setq abbrev
      (intern (pathname-name (first (last (pathname-directory object))))))
    (setq (database-abbreviation dbstruct) abbrev)
    (setq (get abbrev 'abbreviationfor) key)
    (setq (database-operationalist dbstruct) nil)
    (setq (database-operationalist dbstruct)
      (fetchdata alist in "operationAlist")
    (setq (database-constructorform dbstruct) constructorform)
    (setq (database-sourcefile dbstruct) (fetchdata alist in "sourceFile"))
    (when make-database?
      (setq (database-sourcefile dbstruct)
        (file-namestring (database-sourcefile dbstruct)))
    (setq (database-constructorkind dbstruct)
      (setq kind (fetchdata alist in "constructorKind")))
    (setq (database-constructorcategory dbstruct)
      (fetchdata alist in "constructorCategory")
    (setq (database-documentation dbstruct)
      (fetchdata alist in "documentation")
    (setq (database-attributes dbstruct)
      (fetchdata alist in "attributes")
    (setq (database-predicates dbstruct)
      (fetchdata alist in "predicates")
    (setq (database-niladic dbstruct)
      (when (fetchdata alist in "NILASIC") t))
    (addoperations key oldmaps)
    (unless make-database?
      (if (eq kind '|category|)
        (setq (database-ancestors dbstruct)

34.1. DATABASE STRUCTURE

(sublisis |$FormalMapVariableList|
  (cdr constructorform) (fetchdata alist in "ancestors")))

(|updateDatabase| key key systemdirp) ; makes many hashtables???
(|installConstructor| key kind) ; used to be key cname ...
(|updateCategoryTable| key kind)
(if |$InteractiveMode| (setq |$CategoryFrame| |$EmptyEnvironment|)))
(setf (database-cosig dbstruct)
  (cons nil (mapcar #'|categoryForm?|
    (cddar (database-constructormodemap dbstruct))))))

(remprop key 'loaded)
(if (null noexpose) (|setExposeAddConstr| (cons key nil)))
(setf (symbol-function key) ; sets the autoload property for cname
  #'(lambda (&rest args)
    (unless (get key 'loaded)
      (|startTimingProcess| '|load|)
      (|loadLibNoUpdate| key key object)) ; used to be cname key
    (apply key args)))
(|sayKeyedMsg| "%1 will be automatically loaded when needed from %2"
  (list key object))))))

---

defun updateDatabase

For now in NRUNTIME do database update only if forced [constructor? p??]
[cleanClams p??]
[cleanAllSlams p??]
|$forceDatabaseUpdate p??|

— defun updateDatabase —

(defun |updateDatabase| (fname cname systemdirp)
  (declare (ignore fname))
  (declare (special |$forceDatabaseUpdate|))
  (when |$forceDatabaseUpdate|
    (when (constructor? cname)
      (|clearClams|)
      (|clearAllSlams| nil)
      (when (getl cname 'loaded) (|clearConstructorCaches|)))
    (when (or |$forceDatabaseUpdate| (null systemdirp))
      (|clearClams|)
      (|clearAllSlams| nil))))

---
defvar *sourcefiles*

—— initvars ——

(defun Make new databases

Making new databases consists of:

1. reset all of the system hash tables
2. set up Union, Record and Mapping
3. map )library across all of the system files (fills the databases)
4. loading some normally autoloaded files
5. making some database entries that are computed (like ancestors)
6. writing out the databases
7. write out ‘warm’ data to be loaded into the image at build time

Note that this process should be done in a clean image followed by a rebuild of the system image to include the new index pointers (e.g. *interp-stream-stamp*)

The system will work without a rebuild but it needs to re-read the databases on startup. Rebuilding the system will cache the information into the image and the databases are opened but not read, saving considerable startup time. Also note that the order the databases are written out is critical. The interp.daase depends on prior computations and has to be written out last.

The build-name-to-pamphlet-hash builds a hash table whose key-~value is:

- abbreviation ~> pamphlet file name
- abbreviation-line ~> pamphlet file position
- constructor ~> pamphlet file name
- constructor-line ~> pamphlet file position

is the symbol of the constructor name and whose value is the name of the source file without any path information. We hash the constructor abbreviation to pamphlet file name.
34.1. DATABASE STRUCTURE

(defun make-databases (ext dirlist)
  (labels ((build-name-to-pamphlet-hash (dir)
            (let ((ht (make-hash-table) (eof '(done)) point mark abbrev name file ns)
              (dolist (fn (directory dir))
                (when (and (string= (pathname-type fn) "pamphlet")
                            (or (string= (pathname-name fn) "bookvol10.2") ; category
                                 (string= (pathname-name fn) "bookvol10.3") ; domain
                                 (string= (pathname-name fn) "bookvol10.4") ; package
                                 (string= (pathname-name fn) "bookvol10.5")))
                  (with-open-file (f fn)
                    (do ((ln (read-line f nil eof) (read-line f nil eof))
                        (line 0 (incf line)))
                      ((eq ln eof))
                      (when (and (setq mark (search ")abb" ln)) (= mark 0))
                        (setq mark (position \\space ln :from-end t)))
                    (setq name (intern (string-trim \\space (subseq ln mark)))))
                  (cond
                    ((setq mark (search "domain" ln)) (setq mark (+ mark 7)))
                    ((setq mark (search "package" ln)) (setq mark (+ mark 8)))
                    ((setq mark (search "category" ln)) (setq mark (+ mark 9)))
                    (setq point (position \\space ln :start (+ mark 1))))))
              ht))
      (labels ((build-name-to-pamphlet-hash (dir)
                  (let ((ht (make-hash-table) (eof '(done)) point mark abbrev name file ns)
                    (dolist (fn (directory dir))
                      (when (and (string= (pathname-type fn) "pamphlet")
                                  (or (string= (pathname-name fn) "bookvol10.2") ; category
                                       (string= (pathname-name fn) "bookvol10.3") ; domain
                                       (string= (pathname-name fn) "bookvol10.4") ; package
                                       (string= (pathname-name fn) "bookvol10.5")))
                        (with-open-file (f fn)
                          (do ((ln (read-line f nil eof) (read-line f nil eof))
                              (line 0 (incf line)))
                            ((eq ln eof))
                            (when (and (setq mark (search ")abb" ln)) (= mark 0))
                              (setq mark (position \\space ln :from-end t)))
                          (setq name (intern (string-trim \\space (subseq ln mark)))))
                        (cond
                          ((setq mark (search "domain" ln)) (setq mark (+ mark 7)))
                          ((setq mark (search "package" ln)) (setq mark (+ mark 8)))
                          ((setq mark (search "category" ln)) (setq mark (+ mark 9)))
                          (setq point (position \\space ln :start (+ mark 1))))))
                  ht))))
(setq abbrev
  (intern (string-trim '(#\space) (subseq ln mark point))))
(setq ns (namestring fn))
(setq mark (position #\/ ns :from-end t))
(setq file (subseq ns (+ mark 1))))
(setq (gethash abbrev ht) file)
(setq (gethash (format nil "~a-line" abbrev) ht) line)
(setq (gethash name ht) file)
(setq (gethash (format nil "~a-line" name) ht) line))))
ht))

;; these are types which have no library object associated with them.
;; we store some constructed data to make them perform like library
;; objects, the *operationalist-hash* key entry is used by allConstructors
(withSpecialConstructors ()
  (declare (special *allconstructors*))
  ; note: if item is not in *operationalist-hash* it will not be written
  ; Category
  (setf (get '|Category| 'database)
    (make-database :operationalist nil :niladic t))
  (push '|Category| *allconstructors*)
  ; UNION
  (setf (get '|Union| 'database)
    (make-database :operationalist nil :constructorkind '|domain|))
  (push '|Union| *allconstructors*)
  ; RECORD
  (setf (get '|Record| 'database)
    (make-database :operationalist nil :constructorkind '|domain|))
  (push '|Record| *allconstructors*)
  ; MAPPING
  (setf (get '|Mapping| 'database)
    (make-database :operationalist nil :constructorkind '|domain|))
  (push '|Mapping| *allconstructors*)
  ; ENUMERATION
  (setf (get '|Enumeration| 'database)
    (make-database :operationalist nil :constructorkind '|domain|))
  (push '|Enumeration| *allconstructors*)
)

(final-name (root)
  (format nil "~a.daase~a" root ext))
)
(let (d)
  (declare (special |$constructorList| *sourcefiles*
    *allconstructors* *operation-hash*))
  (do-symbols (symbol)
    (when (get symbol 'database)
      (setf (get symbol 'database) nil)))
  (setq *hascategory-hash* (make-hash-table :test #'equal))
  (setq *operation-hash* (make-hash-table))
  (setq *allconstructors* nil)
  (withSpecialConstructors)
34.1. DATABASE STRUCTURE

```
(localdatabase nil
  (list (list '|dir| (namestring (truename "./")) ))
  'make-database)
(dolist (dir dirlist)
  (localdatabase nil
    (list (list '|dir| (namestring (truename (format nil "/~a" dir))))
      'make-database)))

;browse.daase
(load (concatenate 'string (getenviron "AXIOM") "/autoload/topics")) ;; hack
(|browserAutoloadOnceTrigger|)
(|mkTopicHashTable|)
(setq |constructorList| nil) ;; affects buildLibdb
(setq *sourcefiles* (build-name-to-pamphlet-hash
  (concatenate 'string (getenviron "AXIOM") "../../books/*.pamphlet")))
(|buildLibdb|)
(|dbSplitLibdb|)

|dbAugmentConstructorDataTable|
(|mkUsersHashTable|)
(|saveUsersHashTable|)
(|mkDependentsHashTable|)
(|saveDependentsHashTable|)

; (|buildGloss|)
|write-browsedb|
|write-operationdb|

; note: genCategoryTable creates a new *hascategory-hash* table
; this smashes the existing table and regenerates it.
; |write-categorydb| does getdatabase calls to write the new information

|write-categorydb|
(dolist (con (|allConstructors|))
  (let (dbstruct)
    (when (setq dbstruct (get con 'database))
      (setf (database-cosig dbstruct)
        (cons nil (mapcar #'|categoryForm?|
          (cddar (database-constructormodemap dbstruct))))))
    (when (and (|categoryForm?| con)
      (= (length (setq d (|domainsOf| (list con) NIL NIL))) 1))
      (setq d (caar d))
      (when (= (length d) (length (|getConstructorForm| con)))
        (format t " ~a has a default domain of "a"%" con (car d))
        (setf (database-defaultdomain dbstruct) (car d)))))

; note: genCategoryTable creates *ancestors-hash*. |write-interpdb|
; does gethash calls into it rather than doing a getdatabase call.

|write-interpdb|
|write-warmdata|
```

```
(when (probe-file (final-name "interp"))
  (delete-file (final-name "interp")))
(rename-file "interp.build" (final-name "interp"))
(when (probe-file (final-name "operation"))
  (delete-file (final-name "operation")))
(rename-file "operation.build" (final-name "operation"))
```
(when (probe-file (final-name "browse"))
  (delete-file (final-name "browse")))
(rename-file "browse.build"
  (final-name "browse"))
(when (probe-file (final-name "category"))
  (delete-file (final-name "category")))
(rename-file "category.build"
  (final-name "category")))

— defun saveDependentsHashTable —
(defun |saveDependentsHashTable| ()
  (let (stream)
    (declare (special |$depTb| |$erase|))
    (setq stream (|writeLib1| |dependents| 'database 'a|))
    (dolist (k (msort (hkeys |$depTb|)))
      (|rwrite| k (hget |$depTb| k) stream))
    (rshut stream)))

— defun saveUsersHashTable —
(defun |saveUsersHashTable| ()
  (let (stream)
    (declare (special |$depTb| |$erase|))
    (setq stream (|writeLib1| |dependents| 'database 'a|))
    (dolist (k (msort (hkeys |$depTb|)))
      (|rwrite| k (hget |$depTb| k) stream))
    (rshut stream)))

— defun saveDependentsHashTable —
(defun |saveDependentsHashTable| ()
  (let (stream)
    (declare (special |$depTb| |$erase|))
    (setq stream (|writeLib1| |dependents| 'database 'a|))
    (dolist (k (msort (hkeys |$depTb|)))
      (|rwrite| k (hget |$depTb| k) stream))
    (rshut stream)))

— defun saveUsersHashTable —
(defun |saveUsersHashTable| ()
  (let (stream)
    (declare (special |$depTb| |$erase|))
    (setq stream (|writeLib1| |dependents| 'database 'a|))
    (dolist (k (msort (hkeys |$depTb|)))
      (|rwrite| k (hget |$depTb| k) stream))
    (rshut stream)))
34.1. DATABASE STRUCTURE

--- defun saveUsersHashTable ---

(defun |saveUsersHashTable| ()
  (let (stream)
    (declare (special |$usersTb| $erase))
    ($erase '|users| 'database '|a|)
    (setq stream (|writeLib1| '|users| 'database '|a|))
    (dolist (k (msort (hkeys |$usersTb|)))
      (|rwrite| k (HGET |$usersTb| k) stream))
    (rshut stream)))

---

defun Construct the proper database full pathname

--- defun DaaseName ---

(defun DaaseName (name erase?)
  (let (daase filename)
    (declare (special $spadroot))
    (if (setq daase (getenviron "DAASE"))
      (progn
        (setq filename (concatenate 'string daase "/algebra/" name))
        (format t " Using local database ~a.." filename))
    (setq filename (concatenate 'string $spadroot "/algebra/" name)))
    (when erase? (system::system (concatenate 'string "rm -f " filename)))
    filename))

---

Building the interp.daase from hash tables

format of an entry in interp.daase:
(constructor-name
  operationalist
  constructormodemap
  modemaps -- this should not be needed. eliminate it.
  object -- the name of the object file to load for this con.
constructorcategory -- note that this info is the cadar of the
constructormodemap for domains and packages so it is stored
as NIL for them. it is valid for categories.
niladic -- t or nil directly
unused
cosig -- kept directly
constructorkind -- kept directly
defaultdomain -- a short list, for %i
ancestors -- used to compute new category updates
)

Here I'll try to outline the interp database write procedure

(defun write-interpdb ()
"build interp.daase from hash tables"
(declare (special $spadroot *ancestors-hash*))
(let (opalistpos modemapspos cmodemappos master masterpos obj *print-pretty*
concategory categorypos kind niladic cosig abbrev defaultdomain
ancestors ancestorspos out)
(declare (special *print-pretty*))
(print "building interp.daase")

; 1. We open the file we're going to create
(setq out (open "interp.build" :direction :output))

; 2. We reserve some space at the top of the file for the key-time pair
; We will overwrite these spaces just before we close the file.
(princ " " out)

; 3. Make sure we write it out
(finish-output out)

; 4. For every constructor in the system we write the parts:
(dolist (constructor (|allConstructors|))
(let (struct)

; 4a. Each constructor has a property list. A property list is a list
; of (key . value) pairs. The property we want is called 'database
; so there is a ('database . something) in the property list
(setq struct (get constructor 'database))

; 5 We write the "operationsalist"
; 5a. We remember the current file position before we write
; We need this information so we can seek to this position on read
(setq opalistpos (file-position out))
34.1. DATABASE STRUCTURE

; 5b. We get the "operationalist" and write it out
   (print (database-operationalist struct) out)
; 5c. We make sure it was written
   (finish-output out)

; 6 We write the "constructormodemap"
; 6a. We remember the current file position before we write
   (setq cmodemappos (file-position out))
; 6b. We get the "constructormodemap" and write it out
   (print (database-constructormodemap struct) out)
; 6c. We make sure it was written
   (finish-output out)

; 7. We write the "modemaps"
; 7a. We remember the current file position before we write
   (setq modemapspos (file-position out))
; 7b. We get the "modemaps" and write it out
   (print (database-modemaps struct) out)
; 7c. We make sure it was written
   (finish-output out)

; 8. We remember source file pathnames in the obj variable
   (setq obj
      (pathname-name
         (first (last (pathname-directory (database-object struct))))))

; 9. We write the "constructorcategory", if it is a category, else nil
; 9a. Get the constructorcategory
   (setq concategory (database-constructorcategory struct))
; 9b. If we have any data we write it out, else we don’t write it
    Note that if there is no data then the byte index for the
    constructorcategory will not be a number but will be nil.
(if concategory ; if category then write data else write nil
 (progn
   (setq categorypos (file-position out))
   (print concategory out)
   (finish-output out))
   (setq categorypos nil))

; 10. We get a set of properties which are kept as "immediate" data
; 10a. niladic data
   (setq niladic (database-niladic struct))

; 10b. abbreviation data (e.g. POLY for polynomial)
   (setq abbrev (database-abbreviation struct))

; 10c. cosig data
   (setq cosig (database-cosig struct))

; 10d. kind data
   (setq kind (database-constructorkind struct))

; 10e. defaultdomain data
   (setq defaultdomain (database-defaultdomain struct))

; 11. The ancestor data might exist. If it does we fetch it
; 11a. and write it out. If it does not we place
; 11b. and immediate value of nil in the key-value table
   (setq ancestors (gethash constructor *ancestors-hash*)) ;cattable.boot
   (if ancestors
   (progn
     (setq ancestorspos (file-position out))
     (print ancestors out)
     (finish-output out))
     (setq ancestorspos nil))

; 12. "master" is an alist. Each element of the alist has the name of
; 12a. the constructor and all of the above attributes. When the loop
; 12b. finishes we will have constructed all of the data for the key-value
; 12c. table
   (push (list constructor opalistpos cmodemappos modemapspos
            obj categorypos niladic abbrev cosig kind defaultdomain
            ancestorspos) master)))
34.1. DATABASE STRUCTURE

; 13. The loop is done, we make sure all of the data is written

(finish-output out)

; 14. We remember where the key-value table will be written in the file

(setq masterpos (file-position out))

; 15. We print the key-value table

(print master out)

; 16. We make sure we write the table

(finish-output out)

; 17. We go to the top of the file

(file-position out 0)

; 18. We write out the (master-byte-position . universal-time) pair
 ; Note that if the universal-time value matches the value of
 ; *interp-stream-stamp* then there is no reason to read the
 ; interp database because all of the data is already cached in
 ; the image. This happens if you build a database and immediately
 ; save the image. The saved image already has the data since we
 ; just wrote it out. If the *interp-stream-stamp* and the database
 ; time stamp differ we "reread" the database on startup. Actually
 ; we just open the database and fetch as needed. You can see fetches
 ; by setting the *miss* variable non-nil.

(print (cons masterpos (get-universal-time)) out)

; 19. We make sure we write it.

(finish-output out)

; 20 And we are done

(close out))

defun Write the interp database

[spadroot p133]
[*ancestors-hash* p??]
[*print-pretty* p??]
— defun write-interpdb —

(defun write-interpdb ()
  "build interp.daase from hash tables"
  (declare (special $spadroot *ancestors-hash*))
  (let ((opalistpos modemapspos cmodemappos master masterpos obj *print-pretty*
            concategory categorypos kind niladic cosig abbrev defaultdomain
            ancestors ancestorspos out)
        (declare (special *print-pretty*))
        (print "building interp.daase")
        (setq out (open "interp.build" :direction :output)))
    (princ " " out)
    (finish-output out)
    (dolist (constructor (|allConstructors|))
      (let (struct)
        (setq struct (get constructor 'database))
        (setq opalistpos (file-position out))
        (print (database-operationalist struct) out)
        (finish-output out)
        (setq cmodemappos (file-position out))
        (print (database-constructormodemap struct) out)
        (finish-output out)
        (setq modemapspos (file-position out))
        (print (database-modemaps struct) out)
        (finish-output out)
        (setq obj
          (pathname-name
            (first (last (pathname-directory (database-object struct))))))
        (setq concategory (database-constructorcategory struct))
        (if concategory ; if category then write data else write nil
          (progn
            (setq categorypos (file-position out))
            (print concategory out)
            (finish-output out))
          (setq categorypos nil))
        (setq niladic (database-niladic struct))
        (setq abbrev (database-abbreviation struct))
        (setq cosig (database-cosig struct))
        (setq kind (database-constructorkind struct))
        (setq defaultdomain (database-defaultdomain struct))
        (setq ancestors (gethash constructor *ancestors-hash*)) ;cattable.boot
        (if ancestors
          (progn
            (setq ancestorspos (file-position out))
            (print ancestors out)
            (finish-output out))
          (setq ancestorspos nil))
        (push (list constructor opalistpos cmodemappos modemapspos
                  obj categorypos niladic abbrev cosig kind defaultdomain
                  ancestorspos) master)))
Building the browse.daase from hash tables

format of an entry in browse.daase:
( constructorname
  sourcefile
  constructorform
  documentation
  attributes
  predicates
)

This is essentially the same overall process as write-interpdb.
We reserve some space for the (key-table-byte-position . timestamp)
We loop across the list of constructors dumping the data and remembering the byte positions
in a key-value pair table.
We dump the final key-value pair table, write the byte position and time stamp at the top
of the file and close the file.

(defun Write the browse database

[allConstructors p1178]
[$spadroot p133]
[*sourcefiles* p1164]
[*print-pretty* p?]  

;; defun write-browsedb —

(defun write-browsedb ()
  "make browse.daase from hash tables"
  (declare (special $spadroot *sourcefiles*))
  (let (master masterpos src formpos docpos attpos predpos *print-pretty* out)
    (declare (special *print-pretty*))
    (print "building browse.daase")
Building the category.daase from hash tables

This is a single table of category hash table information, dumped in the database format.

defun Write the category database

[genCategoryTable p??]
[*print-pretty* p??]
[*hasCategory-hash* p1143]

— defun write-categorydb —

(defun write-categorydb ()
  "make category.daase from scratch. contains the *hasCategory-hash* table"

Building the operation.daase from hash tables

This is a single table of operations hash table information, dumped in the database format.

**defun Write the operations database**

[*operation-hash* p1143]  

— defun write-operationdb —

(defun write-operationdb ()  
(let (pos master out)  
(declare (special leaves *operation-hash*))  
(setq out (open "operation.build" :direction :output))  
(princ " " out)  
(finish-output out)  
(maphash #'(lambda (key value)  
(if (or (null value) (eq value t))  
(setq pos value)  
(progn  
(setq pos (file-position out))  
(print value out)  
(finish-output out)))  
(push (list key pos) master)  
*hasCategory-hash*)  
(setq pos (file-position out))  
(print master out)  
(finish-output out)  
(file-position out 0)  
(print (cons pos (get-universal-time)) out)  
(finish-output out)  
(close out)))
(push (cons key pos) master))
*operation-hash*)
(finish-output out)
(setq pos (file-position out))
(print master out)
(file-position out 0)
(print (cons pos (get-universal-time)) out)
(finish-output out)
(close out))

Database support operations

defun Data preloaded into the image at build time

[*topicHash p??]

— defun write-warmdata —

(defun write-warmdata ()
"write out information to be loaded into the image at build time"
(declare (special |$topicHash|))
(with-open-file (out "warm.data" :direction :output)
(format out "(in-package "BOOT")~%")
(format out "((setq |$topicHash| (make-hash-table))~%")
(maphash #'(lambda (k v)
(format out "("gethash '|'~a| |$topicHash|) "a" k v) |$topicHash|)~%"))

— defun allConstructors —

(defun |allConstructors| ()
(declare (special |allconstructors|))
|allconstructors|)
defun Return all operations

[*allOperations* p1146]
[*operation-hash* p1143]

--- defun allOperations ---

(defun |allOperations| ()
  (declare (special *allOperations* *operation-hash*))
  (unless *allOperations*
    (maphash #'(lambda (k v) (declare (ignore v)) (push k *allOperations*))
      *operation-hash*)))
*allOperations*
Chapter 35

System Statistics

defun statisticsInitialization
[gbctime p??]

(defun |statisticsInitialization| ()
 "initialize the garbage collection timer"
 #+:akcl (system:gbctime 0)
 nil)

35.1 Lisp Library Handling

defun loadLib
[startTimingProcess p??]
[getdatabase p1156]
[isSystemDirectory p1183]
[pathnameDirectory p1192]
[loadLibNoUpdate p1183]
[sayKeyedMsg p27]
[namestring p1190]
[clearConstructorCache p??]
[updateDatabase p1163]
[installConstructor p??]
[updateCategoryTable p??]

1181
(defun |loadLib| (cname)
  (let (fullLibName systemdir? update? kind u sig coSig)
    (declare (special |$CategoryFrame| |$InteractiveMode| |$printLoadMsgs|
               |$forceDatabaseUpdate|))
    (|startTimingProcess| `'load|)
    (when (setq fullLibName (getdatabase cname 'object))
      (setq systemdir? (|isSystemDirectory| (|pathnameDirectory| fullLibName))
           update? (or |$forceDatabaseUpdate| (null systemdir?)))
    (cond
      ((null update?) (|loadLibNoUpdate| cname cname fullLibName))
      (t
       (setq kind (getdatabase cname 'constructorkind))
       (when |$printLoadMsgs|
         (|sayKeyedMsg| "Loading %1 for %2 %3"
                      (list (|namestring| fullLibName) kind cname))
       (load fullLibName)
       (|clearConstructorCache| cname)
       (|updateDatabase| cname cname systemdir?)
       (|installConstructor| cname kind)
       (setq u (getdatabase cname 'constructormodemap))
       (|updateCategoryTable| cname kind)
       (setq coSig
         (when u
           (setq sig (cdar u))
           (cons nil (loop for x in (cdr sig) collect (|categoryForm?| x))))))
     (if (null (cdr (getdatabase cname 'constructorform)))
       (setf (get cname 'niladic) t)
       (remprop cname 'niladic))
     (setf (get cname 'loaded) fullLibName)
     (when |$InteractiveMode| (setq |$CategoryFrame| (list (list nil)))
           (|stopTimingProcess| `'load|) t))))
defun isSystemDirectory

(defun isSystemDirectory (dir)
  (declare (special $spadroot))
  (every #'char= $spadroot dir))

defun loadLibNoUpdate

(defun loadLibNoUpdate (cname libName fullLibName)
  (declare (ignore libName))
  (let (kind)
    (declare (special |$CategoryFrame| |$InteractiveMode| |$printLoadMsgs|))
    (setq kind (getdatabase cname 'constructorkind))
    (when |$printLoadMsgs|
      (|sayKeyedMsg| "Loading %1 for %2 %3"
        (list (|namestring| fullLibName) kind cname)))
    (cond
      ((equal (catch 'versioncheck (load fullLibName)) (- 1))
        (princ " wrong library version...recompile ")
        (princ fullLibName)
        (terpri)
        (toplevel))
      (t
        (|clearConstructorCache| cname)
        (|installConstructor| cname kind)
        (setf (get cname 'loaded) fullLibName)
        (when |$InteractiveMode| (setq |$CategoryFrame| (list (list nil))))))
defun loadFunctor

(loadFunctor p
(loadLibIfNotLoaded p)

— defun loadFunctor —

(defun |loadFunctor| (u)
  (cond
    ((null (atom u)) (|loadFunctor| (car u)))
    (t
      (|loadLibIfNotLoaded| u)
      u)))
Chapter 36

Special Lisp Functions

defun compiledLookup

[isDomain p??]
[NRTevalDomain p1189]

— defun compiledLookup —

(defun |compiledLookup| (op sig dollar)
 (setq dollar (|NRTevalDomain| dollar))
 (|basicLookup| op sig dollar dollar))

—-

defmacro hashCode?

— defmacro hashCode? 0 —

(defun |hashCode?| (op sig dollar)
 (setq dollar (|NRTevalDomain| dollar))
 (|basicLookup| op sig dollar dollar))

—-

defun basicLookup

[spadcall p??]
[hashCode? p1185]
--- defun basicLookup ---

(defun basicLookup (op sig domain dollar)
  (let (hashPercent box dispatch lookupFun hashSig val boxval)
    (declare (special $hashSeg $hashOpSet $hashOpApply $hashOp0 $hashOp1))
    (cond
      ((vecp domain)
       (if (isNewWorldDomain domain)
           (oldCompLookup op sig domain dollar)
           (lookupInDomainVector op sig domain dollar))
       (t
        (setq hashPercent
          (if (vecp dollar)
              (hashType (elt dollar 0) 0)
              (hashType dollar 0)))
        (setq box (cons nil nil))
        (cond
          ((null (vecp (setq dispatch (car domain))))
           (error '|bad domain format))
          (t
           (setq lookupFun (elt dispatch 3))
           (cond
             ((eql (elt dispatch 0) 0)
              (setq hashSig
                (cond
                  ((hashCode? sig) sig)
                  ((opIsHasCat op) (hashType sig hashPercent))
                  (t (hashType (cons '|Mapping sig) hashPercent))))
                (when (symbolp op)
                  (cond
                    ((eq op '|Zero) (setq op $hashOp0))
                    ((eq op '|One) (setq op $hashOp1)))))
             (t)
             ((eq dispatch 0) (setq dispatch 1))
             (t)))
        (when (null dispatch)
          (error 'no dispatch))
        (t)
        (setq dispatch (elt dispatch 0)))
    (t)
    (let ((dispatch dispatch))
      (when (null dispatch)
        (error 'no dispatch))
      (t)
      (setq dispatch (elt dispatch 0)))
    (when (null dispatch)
      (error 'no dispatch))
    (t)
    (setq dispatch (elt dispatch 0)))
  (t)
  (let ((dispatch dispatch))
    (when (null dispatch)
      (error 'no dispatch))
    (t)
    (setq dispatch (elt dispatch 0)))
  (when (null dispatch)
    (error 'no dispatch))
  (t)
  (setq dispatch (elt dispatch 0)))

(defun basicLookup (op sig domain dollar)
  (let (hashPercent box dispatch lookupFun hashSig val boxval)
    (declare (special $hashSeg $hashOpSet $hashOpApply $hashOp0 $hashOp1))
    (cond
      ((vecp domain)
       (if (isNewWorldDomain domain)
           (oldCompLookup op sig domain dollar)
           (lookupInDomainVector op sig domain dollar))
       (t
        (setq hashPercent
          (if (vecp dollar)
              (hashType (elt dollar 0) 0)
              (hashType dollar 0)))
        (setq box (cons nil nil))
        (cond
          ((null (vecp (setq dispatch (car domain))))
           (error '|bad domain format))
          (t
           (setq lookupFun (elt dispatch 3))
           (cond
             ((eql (elt dispatch 0) 0)
              (setq hashSig
                (cond
                  ((hashCode? sig) sig)
                  ((opIsHasCat op) (hashType sig hashPercent))
                  (t (hashType (cons '|Mapping sig) hashPercent))))
                (when (symbolp op)
                  (cond
                    ((eq op '|Zero) (setq op $hashOp0))
                    ((eq op '|One) (setq op $hashOp1)))))
             (t)
             ((eq dispatch 0) (setq dispatch 1))
             (t)))
        (when (null dispatch)
          (error 'no dispatch))
        (t)
        (setq dispatch (elt dispatch 0)))
    (t)
    (let ((dispatch dispatch))
      (when (null dispatch)
        (error 'no dispatch))
      (t)
      (setq dispatch (elt dispatch 0)))
  (t)
  (let ((dispatch dispatch))
    (when (null dispatch)
      (error 'no dispatch))
    (t)
    (setq dispatch (elt dispatch 0)))
  (when (null dispatch)
    (error 'no dispatch))
  (t)
  (setq dispatch (elt dispatch 0)))

(defun basicLookup (op sig domain dollar)
  (let (hashPercent box dispatch lookupFun hashSig val boxval)
    (declare (special $hashSeg $hashOpSet $hashOpApply $hashOp0 $hashOp1))
    (cond
      ((vecp domain)
       (if (isNewWorldDomain domain)
           (oldCompLookup op sig domain dollar)
           (lookupInDomainVector op sig domain dollar))
       (t
        (setq hashPercent
          (if (vecp dollar)
              (hashType (elt dollar 0) 0)
              (hashType dollar 0)))
        (setq box (cons nil nil))
        (cond
          ((null (vecp (setq dispatch (car domain))))
           (error '|bad domain format))
          (t
           (setq lookupFun (elt dispatch 3))
           (cond
             ((eql (elt dispatch 0) 0)
              (setq hashSig
                (cond
                  ((hashCode? sig) sig)
                  ((opIsHasCat op) (hashType sig hashPercent))
                  (t (hashType (cons '|Mapping sig) hashPercent))))
                (when (symbolp op)
                  (cond
                    ((eq op '|Zero) (setq op $hashOp0))
                    ((eq op '|One) (setq op $hashOp1)))))
             (t)
             ((eq dispatch 0) (setq dispatch 1))
             (t)))
        (when (null dispatch)
          (error 'no dispatch))
        (t)
        (setq dispatch (elt dispatch 0)))
    (t)
    (let ((dispatch dispatch))
      (when (null dispatch)
        (error 'no dispatch))
      (t)
      (setq dispatch (elt dispatch 0)))
  (t)
  (let ((dispatch dispatch))
    (when (null dispatch)
      (error 'no dispatch))
    (t)
    (setq dispatch (elt dispatch 0)))
  (when (null dispatch)
    (error 'no dispatch))
  (t)
  (setq dispatch (elt dispatch 0)))
[((eq op '|elt|) (setq op |$hashOpApply|))
 ((eq op '|setelt|) (setq op |$hashOpSet|))
 (t (setq op (|hashString| (symbol-name op))))))

(cond
 ((setq val
  (car
    (spadcall (cdr domain) dollar op hashSig box nil lookupFun)))
   val)
 ((|hashCode?| sig) nil)
 ((or (> (|#| sig) 1) (|opIsHasCat| op)) nil)
 (setq boxval
  (spadcall (cdr dollar) dollar op
    (|hashType| (car sig) hashPercent)
    box nil lookupFun))
  (cons #'identity (car boxval)))
 (t nil)))

((|opIsHasCat| op) (|HasCategory| domain sig))
 (t
  (when (|hashCode?| op)
    (cond
      ((eql op |$hashOp1|) (setq op '|One|))
      ((eql op |$hashOp0|) (setq op '|Zero|))
      ((eql op |$hashOpApply|) (setq op '|elt|))
      ((eql op |$hashOpSet|) (setq op '|setelt|))
      ((eql op |$hashSeg|) (setq op 'segment))))
    (cond
      ((and (|hashCode?| sig) (eql sig hashPercent))
       (spadcall
         (car (spadcall (cdr dollar) dollar op '($) box nil lookupFun))))
      (t
       (car
        (spadcall (cdr dollar) dollar op sig box nil lookupFun)))))))))))

---

(defun lookupInDomainVector

[basicLookupCheckDefaults p1188]
[spadcall p??]

— defun lookupInDomainVector —

(defun lookupInDomainVector (op sig domain dollar)
  (if (consp domain)
    (|basicLookupCheckDefaults| op sig domain domain)
    (spadcall op sig dollar (elt domain i))))
defun basicLookupCheckDefaults

[vecp p??]
[error p??]
[hashType p??]
[hashCode? p1185]
[hashString p??]
[spadcall p??]
[$lookupDefaults p??]

— defun basicLookupCheckDefaults —

(defun |basicLookupCheckDefaults| (op sig domain dollar)
 (declare (ignore domain))
 (let (box dispatch lookupFun hashPercent hashSig)
      (declare (special |$lookupDefaults|))
      (setq box (cons nil nil))
      (cond
        ((null (vecp (setq dispatch (car dollar))))
         (|error| '|bad domain format|))
        (t
         (setq lookupFun (elt dispatch 3))
         (cond
          ((eql (elt dispatch 0) 0)
           (setq hashPercent
                 (if (vecp dollar)
                     (|hashType| (elt dollar 0) 0)
                     (|hashType| dollar 0)))
           (setq hashSig
                 (if (|hashCode?| sig)
                     sig
                     (|hashType| (cons '|Mapping| sig) hashPercent))
                 (when (symbolp op) (setq op (|hashString| (symbol-name op))))
                 (car (spadcall (cdr dollar) dollar op hashSig
                              box (null |$lookupDefaults|) lookupFun)))
           (t
            (car (spadcall (cdr dollar) dollar op sig box
                          (null |$lookupDefaults|) lookupFun))))))))

|defun basicLookupCheckDefaults |

(declare (ignore domain))
(let (box dispatch lookupFun hashPercent hashSig)
(declare (special |$lookupDefaults|))
(setq box (cons nil nil))
(cond
  ((null (vecp (setq dispatch (car dollar)))))
   (|error| '|bad domain format|))
  (t
   (setq lookupFun (elt dispatch 3))
   (cond
    ((eql (elt dispatch 0) 0)
     (setq hashPercent
          (if (vecp dollar)
              (|hashType| (elt dollar 0) 0)
              (|hashType| dollar 0)))
     (setq hashSig
          (if (|hashCode?| sig)
              sig
              (|hashType| (cons '|Mapping| sig) hashPercent))
          (when (symbolp op) (setq op (|hashString| (symbol-name op))))
          (car (spadcall (cdr dollar) dollar op hashSig
                         box (null |$lookupDefaults|) lookupFun)))
    (t
     (car (spadcall (cdr dollar) dollar op sig box
                      (null |$lookupDefaults|) lookupFun)))))))
36.1. Axiom control structure macros

Axiom used various control structures in the boot code which are not available in Common Lisp. We write some macros here to make the boot to lisp translations easier to read.

defun put

— defun put —

(defun oldCompLookup
  [lookupInDomainVector p1187]
  [lookupDefaults p??]

  — defun oldCompLookup —

  (defun |oldCompLookup| (op sig domvec dollar)
    (let (($lookupDefaults| u)
      (declare (special $lookupDefaults)))
      (setq $lookupDefaults nil)
      (cond
        ((setq u (|lookupInDomainVector| op sig domvec dollar))
          u)
        (t
          (setq $lookupDefaults| t)
          (|lookupInDomainVector| op sig domvec dollar))))

  ————

  defun NRTevalDomain

  [qcar p??]
  [eval p??]
  [evalDomain p1075]

  — defun NRTevalDomain —

  (defun |NRTevalDomain| (form)
    (if (and (consp form) (eq (qcar form) 'setelt))
      (|eval| form)
      (|evalDomain| form)))

  ————

36.1. Axiom control structure macros

Axiom used various control structures in the boot code which are not available in Common Lisp. We write some macros here to make the boot to lisp translations easier to read.

defun put

— defun put —
(defun put (sym ind val) (setf (get sym ind) val))

defmacro while
While the condition is true, repeat the body. When the condition is false, return t.
   — defmacro while —
(defmacro while (condition &rest body)
  '(loop (if (not ,condition) (return t)) ,@body))

defmacro whileWithResult
While the condition is true, repeat the body. When the condition is false, return the result form’s value.
   — defmacro whileWithResult —
(defmacro whileWithResult (condition result &rest body)
  '(loop (if (not ,condition) ,@result) ,@body))

36.2 Filename Handling
This code implements the Common Lisp pathname functions for Lisp/VM. On VM, a filename is 3-list consisting of the filename, filetype and filemode. We also UPCASE everything.

defun namestring
[pathname p1192]
   — defun namestring —
(defun namestring (arg)
  (namestring ( pathname arg)))
36.2. FILENAME HANDLING

(defun pathnameName
  [pathname p1192]
  — defun pathnameName —
  (defun |pathnameName| (arg)
    (pathname-name ([pathname] arg)))

(defun pathnameType
  [pathname p1192]
  — defun pathnameType —
  (defun |pathnameType| (arg)
    (pathname-type ([pathname] arg)))

(defun pathnameTypeId
  [upcase p1206]
  [object2Identifier p228]
  [pathnameType p1191]
  — defun pathnameTypeId —
  (defun |pathnameTypeId| (arg)
    (upcase ([object2Identifier] ([pathnameType] arg))))

(defun mergePathnames
  [pathnameName p1191]
  [pathnameType p1191]
  [pathnameDirectory p1192]
  — defun mergePathnames —
(defun mergePathnames (a b)
  (let (fn ft fm)
    (cond
      ((string= (setq fn (pathnameName a)) ")*") b)
      ((not (equal fn (pathnameName b))) a)
      ((string= (setq ft (pathnameType a)) ")*") b)
      ((not (equal ft (pathnameType b))) a)
      (true (equal (setq fm (pathnameDirectory a)) (list ")*" )) b)
    (t a))))

(defun pathnameDirectory

  (defun pathnameDirectory (arg)
    (namestring (make-pathname :directory (pathname-directory (pathname arg))))))

(defun Axiom pathnames

  (defun pathname (p)
    (cond
      ((null p) p)
      ((pathnamep p) p)
      ((null (consp p)) (pathname p))
      (t
        (when (> (abs p) 2) (setq p (cons (elt p 0) (cons (elt p 1) nil))))
        (pathname (apply #'make-filename p)))))
36.2. FILENAME HANDLING

---

**defun makePathname**

(defun makePathname (name type dir)
  (declare (ignore dir))
  (pathname (list (object2String name) (object2String type))))

---

**defun Delete a file**

(defun deleteFile (arg)
  (declare (special $erase))
  ($erase (pathname arg)))

---

**defun wrap**

(defun wrap (list-of-items wrapper)
  (prog nil
    (cond
      ((or (not (consp list-of-items)) (not wrapper))
        (return list-of-items))
      ((not (consp wrapper))
        (setq wrapper (lotsof wrapper)))
      (return
        (cons
          (if (first wrapper)
          ...
```
'(,(first wrapper) ,(first list-of-items))
(first list-of-items))
(wrap (cdr list-of-items) (cdr wrapper))))
```
(defmacro HGET (table key &rest default) `(gethash ,key ,table ,@default))

---

defun hkeys

---

defun hkeys (table)
(let (keys)
  (maphash #'(lambda (key val) (declare (ignore val)) (push key keys)) table)
  keys))

---

defun digitp

[digitp p1195]

---

defun digitp (x)
(or (and (symbolp x) (digitp (symbol-name x)))
  (and (characterp x) (digit-char-p x))
  (and (stringp x) (= (length x) 1) (digit-char-p (char x 0)))))

---

defun pname

Note it is important that PNAME returns nil not an error for non-symbols

---

defun pname (x)
(cond ((symbolp x) (symbol-name x))
  ((characterp x) (string x))
  (t nil)))

---
defun size

— defun size —

(defun size (l)
  (cond
   ((vectorp l) (length l))
   ((consp l) (list-length l))
   (t 0)))

——

defun strpos

— defun strpos —

(defun strpos (what in start dontcare)
  (setq what (string what) in (string in))
  (if dontcare
      (progn
       (setq dontcare (character dontcare))
       (search what in :start2 start
        :test #'(lambda (x y) (or (eql x dontcare) (eql x y))))))
  (if (= start 0)
    (search what in)
    (search what in :start2 start))))

——

defun strposl

Note that this assumes “table” is a string.

— defun strposl —

(defun strposl (table cvec sint item)
  (setq cvec (string cvec))
  (if (not item)
      (position table cvec :test #'(lambda (x y) (position y x)) :start sint)
    (position table cvec :test-not #'(lambda (x y) (position y x)) :start sint)))

——
defmacro identp

— defmacro identp 0 —

(defun identp (x)
  (if (atom x)
      '(and ,x (symbolp ,x))
      (let ((xx (gensym)))
        '(let ((,xx ,x))
          (and ,xx (symbolp ,xx)))))

---

defun concat

[string-concatenate p??]

— defun concat 0 —

(defun concat (a b &rest l)
  (if (bit-vector-p a)
      (if l
          (apply #'concatenate 'bit-vector a b l)
          (concatenate 'bit-vector a b))
      (if l
          (apply #'system:string-concatenate a b l)
          (system:string-concatenate a b))))

This function was called [functionp] which is a lower-case version of the common lisp function called functionp. Camm Maguire found a bug related to this ambiguity so this was renamed.

defun canFuncall?

— defun canFuncall? —

(defun canFuncall? (fn)
  (if (identp fn)
      (and (fboundp fn) (not (macro-function fn)))
      (functionp fn)))
defun brightprint

[messageprint p1199]

(defun brightprint (x)
  (messageprint x))

---

defun brightprint-0

[messageprint-1 p1199]

(defun brightprint-0 (x)
  (messageprint-1 x))

---

defun member

---

(defun member (item sequence)
  (cond
    ((symbolp item) (member item sequence :test #'eq))
    ((stringp item) (member item sequence :test #'equal))
    ((and (atom item) (not (arrayp item))) (member item sequence))
    (t (member item sequence :test #'equalp))))
36.2. FILENAME HANDLING

defun messageprint
— defun messageprint —
(defun messageprint (x)
(mapc #’messageprint-1 x))

———-

defun messageprint-1
[identp p1197]
[messageprint-1 p1199]
[messageprint-2 p1199]
— defun messageprint-1 —
(defun messageprint-1 (x)
(cond
((or (eq x ’|%l|) (equal x "%l")) (terpri))
((stringp x) (princ x))
((identp x) (princ x))
((atom x) (princ x))
((princ "(")
(messageprint-1 (car x))
(messageprint-2 (cdr x))
(princ ")"))))

———-

defun messageprint-2
[messageprint-1 p1199]
[messageprint-2 p1199]
— defun messageprint-2 —
(defun messageprint-2 (x)
(if (atom x)
(unless x (progn (princ " . ") (messageprint-1 x)))
(progn (princ " ") (messageprint-1 (car x)) (messageprint-2 (cdr x)))))

———-

1199


defun sayBrightly1

[brightprint-0 p1198]
[brightprint p1198]

— defun sayBrightly1 —

(defun sayBrightly1 (x *standard-output*)
  (if (atom x)
      (progn (brightprint-0 x) (terpri) (force-output))
      (progn (brightprint x) (terpri) (force-output)))))

——

defmacro assq

TPDHERE: This could probably be replaced by the default assoc using eql
— defmacro assq —

(defmacro assq (a b)
  `(assoc ,a ,b :test #'eq))

——

defun A version of GET that works with lists

— defun getl 0 —

(defun getl (op prop)
  (when (and op (symbolp op)) (get op prop)))

——
Chapter 37

Record, Union, Mapping, and Enumeration

— postvars —

(eval-when (eval load)
  (mapcar #'(lambda (alist)
                (setf (get (first alist) '|makeFunctionList|) (second alist)))
    '((|Record| |mkRecordFunList|)
      (|Union| |mkUnionFunList|)
      (|Mapping| |mkMappingFunList|)
      (|Enumeration| |mkEnumerationFunList|))))

———
Chapter 38

Common Lisp Algebra Support

These functions are called directly from the algebra source code. They fall into two basic categories, one are the functions that are raw Comon Lisp calls and the other are Axiom specific functions or macros.

Raw function calls are used where there is an alignment of the Axiom type and the underlying representation in Common Lisp. These form the support pillars upon which Axiom rests. For instance, the 'EQ' function is called to support the Axiom equivalent 'eq?' function.

Macros are used to add type information in order to make low level operations faster. An example is the use of macros in DoubleFloat to add Common Lisp type information. Since DoubleFloat is machine arithmetic we give the compiler explicit type information so it can generate fast code.

Functions are used to do manipulations which are Common Lisp operations but the Axiom semantics are not the same. Because Axiom was originally written in Maclisp, then VMLisp, and then Common Lisp some of these old semantics survive.

38.1 AlgebraicFunction

defun retract

[|defun retract|]

1203
(defun |retract| (object)
  (labels (
    (retract1 (object)
      (let (type val underDomain objectp)
        (declare (special |$SingleInteger| |$Integer| |$NonNegativeInteger|
                      |$PositiveInteger|))
        (setq type (|objMode| object))
        (cond
          ((stringp type) '|failed|)
          (t
           (setq val (|objVal| object))
           (cond
            ((equal type |$PositiveInteger|) (mkObj val |$NonNegativeInteger|))
            ((equal type |$NonNegativeInteger|) (mkObj val |$Integer|))
            ((and (equal type |$Integer|) (typep (|unwrap| val) 'fixnum))
             (mkObj val |$SingleInteger|))
            (t
             (cond
              ((or (eql 1 (#| type))
               (and (cons type) (eq (qcar type) '|Union|))
               (and (cons type) (eq (qcar type) '|FunctionCalled|)
                (and (cons (qcdr type)) (eq (qcddr type) nil)))
               (and (cons type) (eq (qcar type) '|OrderedVariableList|
                (and (cons (qcdr type)) (eq (qcddr type) nil)))
               (and (cons type) (eq (qcar type) '|Variable|
                (and (cons (qcdr type)) (eq (qcddr type) nil))))
              (if (setq objectp (|retract2Specialization| object))
                  objectp
                '|failed|)))
            ((null (setq underDomain (|underDomainOf| type)))
             '|failed|)
            (t
             (setq objectp (|retract2Specialization| object))
             objectp
             '|failed|)))
        (t
         (setq objectp (|coerceRetract| object underDomain))
         (cond
          ((eq objectp '|failed|) objectp)
          (t
           (setq objectp (|retract2Specialization| object))
           objectp)))
      (cond
        ((stringp type) '|failed|)
        ((equal type |$EmptyMode|) '|failed|)
        (t
         (setq val (|objVal| object))
         (cond
          ((equal type |$EmptyMode|) '|failed|)
          (t
           (setq objectp (|retract2Specialization| object))
           (cond
            ((eq objectp '|failed|) objectp)
            (t
             (setq val (|objVal| object))
             (cond
              ((equal type |$EmptyMode|) '|failed|)
              (t
               (setq objectp (|retract2Specialization| object)))))))
    (type val ans)
    (declare (special |$EmptyMode|))
    (setq type (|objMode| object))
    (cond
      ((stringp type) '|failed|)
      ((equal type |$EmptyMode|) '|failed|)
      (t
       (setq val (|objVal| object))
       (cond
        ((equal type |$EmptyMode|) '|failed|)
        (t
         (setq objectp (|retract2Specialization| object))
         (cond
          ((equal type |$EmptyMode|) '|failed|)
          (t
           (setq val (|objVal| object))))))))

(Chapter 38. Common Lisp Algebra Support)
38.2 Any

defun spad2BootCoerce

— defun spad2BootCoerce —

(defun |spad2BootCoerce| (x source target)
  (let (xp)
    (cond
      ((null (|isValidType| source))
        (|throwKeyedMsg| "%1p is not a valid type." (list source)))
      ((null (|isValidType| target))
        (|throwKeyedMsg| "%1p is not a valid type." (list target)))
      ((setq xp (|coerceInteractive| (mkObjWrap x source) target))
        (|objValUnwrap| xp))
      (t
        (|throwKeyedMsgCannotCoerceWithValue| (|wrap| x) source target))))

38.3 ApplicationProgramInterface

defun Report what domains get instantiated

— defun reportinstantiations —

(defun reportinstantiations (b)
  (setq |$reportInstantiations| b))
38.4 Boolean

defun The Boolean = function support

— defun BooleanEquality 0 —

(defun |BooleanEquality| (x y) (if x y (null y)))

38.5 Char

defun upcase

[identp p1197]

— defun upcase —

(defun upcase (l)
  (cond ((stringp l) (string-upcase l))
       ((identp l) (intern (string-upcase (symbol-name l))))
       ((characterp l) (char-upcase l))
       ((atom l) l)
       (t (mapcar #'upcase l))))

—

defun downcase

[identp p1197]

— defun downcase —

(defun downcase (l)
  (cond ((stringp l) (string-downcase l))
       ((identp l) (intern (string-downcase (symbol-name l))))
       ((characterp l) (char-downcase l))
       ((atom l) l)
       (t (mapcar #'downcase l))))

—
38.6 ComplexDoubleFloatMatrix

defmacro make-cdouble-matrix

ComplexDoubleFloatMatrix function support
— defmacro make-cdouble-matrix —

(defmacro make-cdouble-matrix (n m)
  '(make-array (list ,n (* 2 ,m)) :element-type 'double-float))

—

defmacro cdaref2

ComplexDoubleFloatMatrix function support
— defmacro cdaref2 —

(defmacro cdaref2 (ov oi oj)
  (let ((v (gensym))
        (i (gensym))
        (j (gensym))
        (v ,ov)
        (i ,oi)
        (j ,oj))
    (cons
     (aref (the (simple-array double-float (* *)) ,v) ,i (* 2 ,j))
     (aref (the (simple-array double-float (* *)) ,v)
           ,i (+ (* 2 ,j) 1)))))

—

defmacro cdsetaref2

ComplexDoubleFloatMatrix function support
— defmacro cdsetaref2 —

(defmacro cdsetaref2 (ov oi oj os)
  (let ((v (gensym))
        (i (gensym))
        (j (gensym))
        (s (gensym))
        (v ,ov)
        (i ,oi)
        (j ,oj))
    (setf
     (aref (the (simple-array double-float (* *)) ,v) ,i (* 2 ,j))
     ,s
     (aref (the (simple-array double-float (* *)) ,v)
           ,i (+ (* 2 ,j) 1))))
defmacro cdanrows

ComplexDoubleFloatMatrix function support

— defmacro cdanrows —

(defmacro cdanrows (v)
  `(array-dimension (the (simple-array double-float (* *)) ,v) 0))

defmacro cdancols

ComplexDoubleFloatMatrix function support

— defmacro cdancols —

(defmacro cdancols (v)
  `(truncate
    (array-dimension (the (simple-array double-float (* *)) ,v) 1) 2))

38.7 ComplexDoubleFloatVector

Complex Double Float Vectors are simple arrays of lisp double-floats made available at the Spad language level. Note that these vectors are 0 based whereas other Spad language vectors are 1-based. Complex array is implemented as an array of doubles. Each complex number occupies two positions in the real array.

defmacro make-cdouble-vector

ComplexDoubleFloatVector Qnew function support

— defmacro make-cdouble-vector —
(defmacro make-cdouble-vector (n)
  '(make-array (list (* 2 ,n)) :element-type 'double-float))

---

defmacro cdelt

ComplexDoubleFloatVector Qelt1 function support
  — defmacro cdelt —

(defmacro CDELT(ov oi)
  (let ((v (gensym))
        (i (gensym)))
    '(let ((,v ,ov)
            (,i ,oi))
      (cons
       (aref (the (simple-array double-float (*)) ,v) (* 2 ,i))
       (aref (the (simple-array double-float (*)) ,v) (+ (* 2 ,i) 1)))))

---

defmacro cdsetelt

ComplexDoubleFloatVector Qasetelt1 function support
  — defmacro cdsetelt —

(defmacro cdsetelt(ov oi os)
  (let ((v (gensym))
        (i (gensym))
        (s (gensym)))
    '(let ((,v ,ov)
            (,i ,oi)
            (,s ,os))
      (setf (aref (the (simple-array double-float (*)) ,v) (* 2 ,i))
            (car ,s))
      (setf (aref (the (simple-array double-float (*)) ,v) (+ (* 2 ,i) 1))
            (cdr ,s))
      ,s)))
defmacro cdlen

ComplexDoubleFloatVector Qsize function support

(defmacro cdlen
  (v)
  '(truncate (length (the (simple-array double-float (*)) ,v)) 2))

38.8 Database

defun Database elt function support

[|basicMatch?| p?]

(defun stringMatches?
  (pattern subject)
  (when (integerp (|basicMatch?| pattern subject)) t))

38.9 DirectProduct

defun vec2list

(defun vec2list (vec) (coerce vec 'list))

38.10 DoubleFloat

These macros wrap their arguments with strong type information in order to optimize doublefloat computations. They are used directly in the DoubleFloat domain (see Volume 10.3).
defmacro DFLessThan
Compute a strongly typed double float comparison See Steele Common Lisp 1990 p293
  — defmacro DFLessThan —
(defmacro DFLessThan (x y)
  '(< (the double-float ,x) (the double-float ,y)))

---

defmacro DFUnaryMinus
Compute a strongly typed unary double float minus See Steele Common Lisp 1990 p295
  — defmacro DFUnaryMinus —
(defmacro DFUnaryMinus (x)
  '(the double-float (- (the double-float ,x))))

---

defmacro DFMinusp
Compute a strongly typed unary double float test for negative See Steele Common Lisp 1990 p292
  — defmacro DFMinusp —
(defmacro DFMinusp (x)
  '(minusp (the double-float ,x)))

---

defmacro DFZerop
Compute a strongly typed unary double float test for zero See Steele Common Lisp 1990 p292
  — defmacro DFZerop —
(defmacro DFZerop (x)
  '(zerop (the double-float ,x)))

---
defmacro DFAdd

Compute a strongly typed doublefloat addition See Steele Common Lisp 1990 p295

(defmacro DFAdd (x y)
  `(the double-float (+ (the double-float ,x) (the double-float ,y))))

---

defmacro DFSubtract

Compute a strongly typed doublefloat subtraction See Steele Common Lisp 1990 p295

(defmacro DFSubtract (x y)
  `(the double-float (- (the double-float ,x) (the double-float ,y))))

---

defmacro DFMultiply

Compute a strongly typed doublefloat multiplication See Steele Common Lisp 1990 p296

(defmacro DFMultiply (x y)
  `(the double-float (* (the double-float ,x) (the double-float ,y))))

---

defmacro DFIntegerMultiply

Compute a strongly typed doublefloat multiplication by an integer. See Steele Common Lisp 1990 p296

(defmacro DFIntegerMultiply (i y)
  `(the double-float (* (the integer ,i) (the double-float ,y))))

---
defmacro DFMax

Choose the maximum of two doublefloats. See Steele Common Lisp 1990 p294
— defmacro DFMax —

(defmacro DFMax (x y)
  '(the double-float (max (the double-float ,x) (the double-float ,y))))

---

defmacro DFMIn

Choose the minimum of two doublefloats. See Steele Common Lisp 1990 p294
— defmacro DFMIn —

(defmacro DFMIn (x y)
  '(the double-float (min (the double-float ,x) (the double-float ,y))))

---

defmacro DFEql

Compare two doublefloats for equality, where equality is eq, or numbers of the same type
with the same value. See Steele Common Lisp 1990 p105
— defmacro DFEql —

(defmacro DFEql (x y)
  '(eql (the double-float ,x) (the double-float ,y)))

---

defmacro DFDivide

Divide a doublefloat by a a doublefloat See Steele Common Lisp 1990 p296
— defmacro DFDivide —

(defmacro DFDivide (x y)
  '(the double-float (/ (the double-float ,x) (the double-float ,y))))

---
defmacro DFIntegerDivide

Divide a doublefloat by an integer See Steele Common Lisp 1990 p296

(defmacro DFIntegerDivide (x i)
  `(the double-float (/ (the double-float ,x) (the integer ,i)))))

---

defmacro DFSqrt

Compute the doublefloat square root of x. The result will be complex if the argument is negative. See Steele Common Lisp 1990 p302

(defmacro DFSqrt (x)
  `(sqrt (the double-float ,x)))

---

defmacro DFLogE

Compute the doublefloat log of x with the base e. The result will be complex if the argument is negative. See Steele Common Lisp 1990 p301

(defmacro DFLogE (x)
  `(log (the double-float ,x)))

---

defmacro DFLog

Compute the doublefloat log of x with a given base b. The result will be complex if x is negative. See Steele Common Lisp 1990 p301

(defmacro DFLog (x b)
  `(log (the double-float ,x) (the fixnum ,b)))

---
defmacro DFIntegerExpt
Compute the doublefloat expt of x with a given integer power i See Steele Common Lisp 1990 p300

(defmacro DFIntegerExpt (x i)
  `(the double-float (expt (the double-float ,x) (the integer ,i)))))

defmacro DFExpt
Compute the doublefloat expt of x with a given power p. The result could be complex if the
base is negative and the power is not an integer. See Steele Common Lisp 1990 p300

(defmacro DFExpt (x p)
  `(expt (the double-float ,x) (the double-float ,p)))

defmacro DFExp
Compute the doublefloat exp with power e See Steele Common Lisp 1990 p300

(defmacro DFExp (x)
  `(the double-float (exp (the double-float ,x))))

defmacro DFSin
Compute a strongly typed doublefloat sin See Steele Common Lisp 1990 p304

(defmacro DFSin (x)
  `(the double-float (sin (the double-float ,x))))
defmacro DFCos

Compute a strongly typed doublefloat cos See Steele Common Lisp 1990 p304

(defmacro DFCos (x)
  `(the double-float (cos (the double-float ,x))))

defmacro DFTan

Compute a strongly typed doublefloat tan See Steele Common Lisp 1990 p304

(defmacro DFTan (x)
  `(the double-float (tan (the double-float ,x))))

defmacro DFAsin

Compute a strongly typed doublefloat asin. The result is complex if the absolute value of
the argument is greater than 1. See Steele Common Lisp 1990 p305

(defmacro DFAsin (x)
  `(asin (the double-float ,x)))

defmacro DFACos

Compute a strongly typed doublefloat acos. The result is complex if the absolute value of
the argument is greater than 1. See Steele Common Lisp 1990 p305

(defmacro DFACos (x)
  `(acos (the double-float ,x)))
defmacro DFAtan

Compute a strongly typed doublefloat atan See Steele Common Lisp 1990 p305

(defmacro DFAtan
  (x)
  '(the double-float (atan (the double-float ,x))))

— defmacro DFAtan —

defmacro DFAtan2

Compute a strongly typed doublefloat atan with 2 arguments

\[
y = 0 \quad x > 0 \quad \text{Positive x-axis} \quad 0 \\
y > 0 \quad x > 0 \quad \text{Quadrant I} \quad 0 < \text{result} < \pi/2 \\
y > 0 \quad x = 0 \quad \text{Positive y-axis} \quad \pi/2 \\
y > 0 \quad x < 0 \quad \text{Quadrant II} \quad \pi/2 < \text{result} < \pi \\
y = 0 \quad x < 0 \quad \text{Negative x-axis} \quad \pi \\
y < 0 \quad x < 0 \quad \text{Quadrant III} \quad -\pi < \text{result} < -\pi/2 \\
y < 0 \quad x = 0 \quad \text{Negative y-axis} \quad -\pi/2 \\
y < 0 \quad x > 0 \quad \text{Quadrant IV} \quad -\pi/2 < \text{result} < 0 \\
y = 0 \quad x = 0 \quad \text{Origin} \quad \text{error}
\]

See Steele Common Lisp 1990 p306

— defmacro DFAtan2 —

(defmacro DFAtan2 (y x)
  '(the double-float (atan (the double-float ,x) (the double-float ,y))))

— defmacro DFAtan2 —

defmacro DFSinh

Compute a strongly typed doublefloat sinh

\[
(e^x - e^{-x})/2
\]

See Steele Common Lisp 1990 p308

— defmacro DFSinh —

(defmacro DFSinh (x)
  '(the double-float (sinh (the double-float ,x))))

— defmacro DFSinh —
defmacro DFCosh
Compute a strongly typed doublefloat cosh

\( \frac{e^z + e^{-z}}{2} \)

See Steele Common Lisp 1990 p308

—— defmacro DFCosh ——

(defmacro DFCosh (x)
  `(the double-float (cosh (the double-float ,x))))

——

defmacro DFTanh
Compute a strongly typed doublefloat tanh

\( \frac{e^z - e^{-z}}{(e^z + e^{-z})} \)

See Steele Common Lisp 1990 p308

—— defmacro DFTanh ——

(defmacro DFTanh (x)
  `(the double-float (tanh (the double-float ,x))))

——

defmacro DFAsinh
Compute the inverse hyperbolic sin.

\( \log \left( z + \sqrt{1 + z^2} \right) \)

See Steele Common Lisp 1990 p308

—— defmacro DFAsinh ——

(defmacro DFAsinh (x)
  `(the double-float (asinh (the double-float ,x))))

——
defmacro DFAcosh

Compute the inverse hyperbolic cos. Note that the acosh function will return a complex result if the argument is less than 1.

\[ \log \left( z + (z + 1)\sqrt{(z - 1)/(z + 1)} \right) \]

See Steele Common Lisp 1990 p308

— defmacro DFAcosh —

(defmacro DFAcosh (x)
 `(acosh (the double-float ,x)))

-------------

defmacro DFAtanh

Compute the inverse hyperbolic tan. Note that the acosh function will return a complex result if the argument is greater than 1.

\[ \log \left( (1 + z)\sqrt{1 / (1 - z^2)} \right) \]

See Steele Common Lisp 1990 p308

— defmacro DFAtanh —

(defmacro DFAtanh (x)
 `(atanh (the double-float ,x)))

-------------

defun Machine specific float numerator

This is used in the DoubleFloat integerDecode function

— defun integer-decode-float-numerator 0 —

(defun integer-decode-float-numerator (x)
 (integer-decode-float x))

-------------
defun Machine specific float denominator

This is used in the DoubleFloat integerDecode function

— defun integer-decode-float-denominator 0 —

(defun integer-decode-float-denominator (x)
  (multiple-value-bind (mantissa exponent sign) (integer-decode-float x)
    (declare (ignore mantissa sign)) (expt 2 (abs exponent))))

defun Machine specific float sign

This is used in the DoubleFloat integerDecode function

— defun integer-decode-float-sign 0 —

(defun integer-decode-float-sign (x)
  (multiple-value-bind (mantissa exponent sign) (integer-decode-float x)
    (declare (ignore mantissa exponent)) sign))

defun Machine specific float bit length

This is used in the DoubleFloat integerDecode function

— defun integer-decode-float-exponent 0 —

(defun integer-decode-float-exponent (x)
  (multiple-value-bind (mantissa exponent sign) (integer-decode-float x)
    (declare (ignore mantissa sign)) exponent))

defun Decode floating-point values

This function is used by DoubleFloat to implement the “mantissa” and “exponent” functions.

— defun manexp 0 —

(defun manexp (u)
  (multiple-value-bind (f e s)
      (decode-float u)
    (cons (* s f) e)))
defun The cotangent routine

The cotangent function is defined as

\[ cot(z) = \frac{1}{\tan(z)} \]

(defun cot 0)

(if (or (> a 1000.0) (< a -1000.0)))
  (/ (cos a) (sin a))
  (/ 1.0 (tan a)))

defun The inverse cotangent function

The inverse cotangent (arc-cotangent) function is defined as

\[ acot(z) = cot^{-1}(z) = tan^{-1}(\frac{1}{z}) \]

See Steele Common Lisp 1990 pp305-307

(defun acot 0)

(if (> a 0.0)
  (if (> a 1.0)
    (atan (/ 1.0 a))
    (- (/ pi 2.0) (atan a)))
  (if (< a -1.0)
    (- pi (atan (/ -1.0 a)))
    (+ (/ pi 2.0) (atan (- a)))))

defun The secant function

\[ sec(x) = \frac{1}{\cos(x)} \]

(defun sec 0)
(defun sec (x) (/ 1 (cos x)))

defun The inverse secant function

\[ asec(x) = \arccos \left( \frac{1}{x} \right) \]

— defun asec 0 —
(defun asec (x) (acos (/ 1 x)))

defun The cosecant function

\[ \csc(x) = \frac{1}{\sin(x)} \]

— defun csc 0 —
(defun csc (x) (/ 1 (sin x)))

defun The inverse cosecant function

\[ \arccsc(x) = \frac{1}{\arcsin(x)} \]

— defun acsc 0 —
(defun acsc (x) (asin (/ 1 x)))
defun The hyperbolic cosecant function

\[ \text{csch}(x) = \frac{1}{\sinh(x)} \]

— defun csch 0 —

(defun csch (x) (/ 1 (sinh x)))

———

defun The hyperbolic cotangent function

\[ \text{coth}(x) = \frac{\cosh(x) \text{csch}(x)}{} \]

— defun coth 0 —

(defun coth (x) (* (cosh x) (csch x)))

———

defun The hyperbolic secant function

\[ \text{sech}(x) = \frac{1}{\cosh(x)} \]

— defun sech 0 —

(defun sech (x) (/ 1 (cosh x)))

———

defun The inverse hyperbolic cosecant function

\[ \text{acsch}(x) = \text{asinh} \left( \frac{1}{x} \right) \]

— defun acsch 0 —
(defun acsch (x) (asinh (/ 1 x)))

---

defun The inverse hyperbolic cotangent function

\[
acoth(x) = \text{atanh} \left( \frac{1}{x} \right)
\]

---

defun acoth 0 ---

(defun acoth (x) (atanh (/ 1 x)))

---

defun The inverse hyperbolic secant function

\[
asech(x) = \text{acosh} \left( \frac{1}{x} \right)
\]

---

defun asech 0 ---

(defun asech (x) (acosh (/ 1 x)))

---

### 38.11 DoubleFloatMatrix

defmacro make-double-matrix

DoubleFloatMatrix qnew function support

---

defmacro make-double-matrix ---

`(defmacro make-double-matrix (n m)
 ' (make-array (list ,n ,m) :element-type 'double-float))

---
defmacro make-double-matrix1

DoubleFloatMatrix new function support
— defmacro make-double-matrix1 —

(defun make-double-matrix1 (n m s)
  `(make-array (list ,n ,m) :element-type 'double-float
               :initial-element ,s))

defmacro daref2

DoubleFloatMatrix qelt function support
— defmacro daref2 —

(defun daref2 (v i j)
  `(aref (the (simple-array double-float (* *)) ,v) ,i ,j))

defmacro dsetaref2

DoubleFloatMatrix qsetelt! function support
— defmacro dsetaref2 —

(defun dsetaref2 (v i j s)
  `(setf (aref (the (simple-array double-float (* *)) ,v) ,i ,j)
         ,s))

defmacro danrows

DoubleFloatMatrix nrows function support
— defmacro danrows —

(defun danrows (v)
  `(array-dimension (the (simple-array double-float (* *)) ,v) 0))
defmacro dancols
DoubleFloatMatrix ncols function support
  — defmacro dancols —

(defmacro dancols (v)
  '(array-dimension (the (simple-array double-float (* *)) ,v) 1))

38.12 DoubleFloatVector

Double Float Vectors are simple arrays of lisp double-floats made available at the Spad language level. Note that these vectors are 0 based whereas other Spad language vectors are 1-based.

defmacro dlen
DoubleFloatVector Qsize function support
  — defmacro dlen —

(defmacro dlen (v)
  '(length (the (simple-array double-float (* *)) ,v)))

defmacro make-double-vector
DoubleFloatVector Qnew function support
  — defmacro make-double-vector —

(defmacro make-double-vector (n)
  '(make-array (list ,n) :element-type 'double-float))

defmacro make-double-vector1
DoubleFloatVector Qnew1 function support
  — defmacro make-double-vector1 —
(defmacro make-double-vector1 (n s)
  '(make-array (list ,n) :element-type 'double-float :initial-element ,s))

---

defmacro delt
DoubleFloatVector Qelt1 function support
  — defmacro delt —

(defmacro delt (v i)
  '(aref (the (simple-array double-float (*)) ,v) ,i))

---

defmacro dsetelt
DoubleFloatVector Qsetelt1 function support
  — defmacro dsetelt —

(defmacro dsetelt (v i s)
  '(setf (aref (the (simple-array double-float (*)) ,v) ,i) ,s))

---

38.13   File

defvar *read-place-holder*

  — initvars —

(defvar *read-place-holder* (make-symbol "%.EOF")
  "default value returned by read and read-line at end-of-file")

---

defun placep

[*read-place-holder* p1227]

  — defun placep 0 —
(defun placep (item)
  (declare (special *read-place-holder*))
  (eq item *read-place-holder*))

(defun vmread
  (defun vmread 0 —
    (funcall vmread (optional (st *standard-input*) (eofval *read-place-holder*))
      (read st nil eofval))

38.14 FileName

defun FileName filename function implementation

[StringToDir p1228]

(defun fnameMake (d n e)
  (if (string= e "") (setq e nil))
  (make-pathname :directory (|StringToDir| d) :name n :type e))

(defun FileName filename support function

[lastc p??]

(defun |StringToDir| (s)
  (cond
    ((string= s "/") ':root)
    ((string= s "") nil)
    (t
      (let ((lastc (aref s (- (length s) 1)))))
(if (char= lastc #\/) (pathname-directory (concat s "name.type")) (pathname-directory (concat s "/name.type")))))}})

defun FileName directory function implementation

- defun fnameDirectory —

(defun |fnameDirectory| (f) (|DirToString| (pathname-directory f)))

defun FileName directory function support

For example, "/" "/u/swatt" "./.src"

- defun DirToString 0 —

(defun |DirToString| (d) (cond ((equal d '(:root)) "/") ((null d) "/") ('t (string-right-trim "/" (namestring (make-pathname :directory d))))) ))

defun FileName name function implementation

- defun fnameName 0 —

(defun |fnameName| (f) (let ((s (pathname-name f))) (if s s "")))

-
defun FileName extension function implementation

— defun fnameType 0 —

(defun fnameType (f)
  (let ((s (pathname-type f)))
    (if s s "") ))

—

defun FileName exists? function implementation

— defun fnameExists? 0 —

(defun fnameExists? (f)
  (if (probe-file (namestring f)) 't nil))

—

defun FileName readable? function implementation

— defun fnameReadable? 0 —

(defun fnameReadable? (f)
  (let ((s (open f :direction :input :if-does-not-exist nil)))
    (cond (s (close s) t) (t nil)))

—

defun FileName writeable? function implementation

[myWriteable? p??]

— defun fnameWritable? —

(defun fnameWritable? (f)
  (myWriteable? (namestring f)) )

—
defun FileName writeable? function support

[error p??]
[fnameExists? p1230]
[fnameDirectory p1229]
[writeablep p??]

— defun myWritable? —

(defun |myWritable?| (s)
  (if (not (stringp s)) (|error| "'myWritable?' requires a string arg."
  (if (string= s "") (setq s ".")
  (if (not (|fnameExists?| s)) (setq s (|fnameDirectory| s)))
  (if (string= s "") (setq s ".")
  (if (> (|writeablep| s) 0) 't nil))

———

defun FileName new function implementation

[fnameMake p1228]

— defun fnameNew —

(defun |fnameNew| (d n e)
  (if (not (|myWritable?| d))
    nil
  (do ((fn))
    (nil)
    (setq fn (|fnameMake| d (string (gensym n)) e))
    (if (not (probe-file (namestring fn)))
      (return-from |fnameNew| fn)))))

———

38.15 IndexedBits

defmacro truth-to-bit

IndexedBits new function support

— defmacro truth-to-bit —

(defmacro truth-to-bit (x) '|(cond ,x 1 ) ('else 0))
defun IndexedBits new function support

— defun bvec-make-full 0 —

(defun bvec-make-full (n x)
  (make-array (list n) :element-type 'bit :initial-element x))

defmacro bit-to-truth

 IndexedBits elt function support
 — defmacro bit-to-truth 0 —

(defmacro bit-to-truth (b) `(eq ,b 1))

defmacro bvec-elt

 IndexedBits elt function support
 — defmacro bvec-elt 0 —

(defmacro bvec-elt (bv i) `(sbit ,bv ,i))

defmacro bvec-setelt

 IndexedBits setelt function support
 — defmacro bvec-setelt —

(defmacro bvec-setelt (bv i x) `(setf (sbit ,bv ,i) ,x))
defmacro bvec-size
IndexedBits length function support
   — defmacro bvec-size —

(defmacro bvec-size (bv) '(size ,bv))

-----
defun IndexedBits concat function support

   — defun bvec-concat 0 —

(defun bvec-concat (bv1 bv2) (concatenate '(vector bit) bv1 bv2))

-----
defun IndexedBits copy function support

   — defun bvec-copy 0 —

(defun bvec-copy (bv) (copy-seq bv))

-----
defun IndexedBits = function support

   — defun bvec-equal 0 —

(defun bvec-equal (bv1 bv2) (equal bv1 bv2))

-----
defun IndexedBits < function support

   — defun bvec-greater 0 —
(defun bvec-greater (bv1 bv2)
  (let ((pos (mismatch bv1 bv2)))
    (cond ((or (null pos) (>= pos (length bv1))) nil)
          ((< pos (length bv2)) (> (bit bv1 pos) (bit bv2 pos)))
          ((find 1 bv1 :start pos) t)
          (t nil)))))

defun IndexedBits And function support

— defun bvec-and 0 —

(defun bvec-and (bv1 bv2) (bit-and bv1 bv2))

defun IndexedBits Or function support

— defun bvec-or 0 —

(defun bvec-or (bv1 bv2) (bit-ior bv1 bv2))

defun IndexedBits xor function support

— defun bvec-xor 0 —

(defun bvec-xor (bv1 bv2) (bit-xor bv1 bv2))

defun IndexedBits nand function support

— defun bvec-nand 0 —
(defun bvec-nand (bv1 bv2) (bit-nand bv1 bv2))

defun IndexedBits nor function support

— defun bvec-nor 0 —

(defun bvec-nor (bv1 bv2) (bit-nor bv1 bv2))

defun IndexedBits not function support

— defun bvec-not 0 —

(defun bvec-not (bv) (bit-not bv))

38.16 IndexCard

defun IndexCard origin function support

[dbPart p??]
[charPosition p??]
[substring p256]

— defun alqlGetOrigin —

(defun |alqlGetOrigin| (x)
  (let (field k)
    (setq field (|dbPart| x 5 1))
    (setq k (|charPosition| #\( field 2))
    (substring field 1 (1- k))))
defun IndexCard origin function support

[dbPart p??]
[charPosition p??]
[substring p256]

— defun alqlGetParams —

(defun alqlGetParams (x)
  (let (field k)
    (setq field (dbPart x 5 1))
    (setq k (charPosition \ field 2))
    (substring field k nil)))

———

defun IndexCard elt function support

[dbPart p??]
[substring p256]

— defun alqlGetKindString —

(defun alqlGetKindString (x)
  (if (or (char= (elt x 0) #\a) (char= (elt x 0) #\o))
    (substring (dbPart x 5 1) 0 1)
    (substring x 0 1)))

———

38.17 IndexedString
defun qenum

This is also used in bookvol10.3 in the CHAR domain.

— defun qenum 0 —

(defun qenum (cvec ind)
  (char-code (char cvec ind)))

———

— defmacro qsize 0 —
(defmacro qsize (x)
  '(the fixnum (length (the simple-string ,x))))

---

— defun qset 0 —

(defun qset (cvec ind charnum)
  (setf (char cvec ind) (code-char charnum)))

---

— defmacro qsgreaterp 0 —

(defmacro qsgreaterp (a b)
  '>(the fixnum ,a) (the fixnum ,b))

---

### 38.18 InputForm

defun called by interpret function

— defun mkObjFn 0 —

(defun mkObjFn (val mode)
  (cons mode val))

---

defun called by interpret function

— defun objValFn 0 —

(defun objValFn (obj)
  (cdr obj))

---
defun called by interpret function

— defun objModeFn 0 —

(defun |objModeFn| (obj)
  (car obj))

——

defun unparseInputForm

This fixes bug 7217. The default title generation is bogus. This is called from the unparse function in InputForm, bookvol10.3 Given a form, \(u\), we try to recover the input line that created it.

[$\text{InteractiveMode} p247$]
[$\text{FormatSigAsTeX} p??$]

— defun unparseInputForm —

(defun |unparseInputForm| (u)
  (let (|$\text{FormatSigAsTeX}$| |$\text{InteractiveMode}$|)
    (declare (special |$\text{FormatSigAsTeX}$| |$\text{InteractiveMode}$|))
    (setq |$\text{FormatSigAsTeX}$| 1)
    (setq |$\text{InteractiveMode}$| nil)
    (|form2StringLocal| u)))

——

38.19 Integer

defun Integer divide function support

Note that this is defined as a SPADReplace function in Integer so that algebra code that uses the Integer divide function actually inlines a call to this code. The Integer domain contains the line:

(PUT (QUOTE |INT;divide;2$R;44|) (QUOTE |SPADreplace|) (QUOTE DIVIDE2))

— defun divide2 0 —
(defun divide2 (x y)
  (multiple-value-call #'cons (truncate x y)))

---

defun Integer quo function support

Note that this is defined as a SPADReplace function in Integer so that algebra code that uses
the Integer quo function actually inlines a call to this code. The Integer domain contains
the line:

(PUT (QUOTE |INT;rem;3$;46|) (QUOTE |SPADreplace|) (QUOTE REMAINDER2))

Because these are identical except for name we make the symbol-functions equivalent. This
was done in the original code for efficiency.

---

defun remainder2 0

(setf (symbol-function 'remainder2) #'rem)

---

defun Integer quo function support

Note that this is defined as a SPADReplace function in Integer so that algebra code that uses
the Integer quo function actually inlines a call to this code. The Integer domain contains
the line:

(PUT (QUOTE |INT;quo;3$;45|) (QUOTE |SPADreplace|) (QUOTE QUOTIENT2))

---

defun quotient2 0

(defun quotient2 (x y)
  (values (truncate x y)))

---

defun Integer random function support

This is used for calls to random with no arguments. If an argument is supplied to random
then the common lisp random function is called directly. This could be lifted up into the
spad code.

---

defun random 0
(defun random () (random (expt 2 26)))

---

38.20 KeyedAccessFile

defun KeyedAccessFile defstream function support

This is a simpler interface to RDEFIOSTREAM [rdefiostream p??]

---

(defun rdefinstream (&rest fn)
  ;; following line prevents rdefiostream from adding a default filetype
  (unless (rest fn) (setq fn (list (pathname (car fn)))))
  (rdefiostream (list (cons 'file fn) '(mode . input))))

---

(defun KeyedAccessFile defstream function support

[rdefiostream p??]

---

(defun rdefoutstream (&rest fn)
  ;; following line prevents rdefiostream from adding a default filetype
  (unless (rest fn) (setq fn (list (pathname (car fn)))))
  (rdefiostream (list (cons 'FILE fn) '(mode . OUTPUT))))

---

38.21 NumberFormats

defun ncParseFromString

---

(defun ncParseFromString (s)
  (zeroOneTran (catch 'SPAD_READER (parseFromString s))))
38.22 OperationsQuery

defun OperationQuery getDatabase function support

This function, called as getBrowseDatabase(arg) returns a list of appropriate entries in the browser database. The legal values for arg are

- “o” (operations)
- “k” (constructors)
- “d” (domains)
- “c” (categories)
- “p” (packages)

(defun |getBrowseDatabase| |kind|)
(let (|$includeUnexposed?|)
(declare (special |$includeUnexposed?|))
(setq |$includeUnexposed?| t)
(when (|member| |kind| ’("o" "k" "c" "d" "p"))
(|grepConstruct| "*" (intern |kind|))))

38.23 ParametricLinearEquations

defun algCoerceInteractive

(defun |algCoerceInteractive| |p| |source| |target|
(let (|$useConvertForCoercions|)
(declare (special |$useConvertForCoercions|)))
(setq $useConvertForCoercions| t)
(setq source (|devaluate| source))
(setq target (|devaluate| target))
(setq u (|coerceInteractive| (mkObjWrap p source) target))
(if u
 (|objValUnwrap| u)
 (|error| (list "can't convert" p "of mode" source "to mode" target))))

### 38.24 Plot3d

We catch numeric errors and throw a different failure than normal. The trapNumericErrors macro will return a pair of the form Union(type-of-form, "failed"). This pair is tested for eq-ness so it has to be unique. It lives in the defvar $numericFailure. The old value of the $BreakMode variable is saved in a defvar named $oldBreakMode.

**defvar $numericFailure**

This is a failed union branch which is the value returned for numeric failure.

— initvars —

(defvar |$numericFailure| (cons 1 "failed"))

———

**defvar $oldBreakMode**

— initvars —

(defvar |$oldBreakMode| nil "the old value of the $BreakMode variable")

———

**defmacro trapNumericErrors**

The following macro evaluates form returning Union(type-of form, "failed"). It is used in the myTrap local function in Plot3d.

— defmacro trapNumericErrors —
(defmacro |trapNumericErrors| (form)
  `(let ((|$oldBreakMode| |$BreakMode|) (|$BreakMode| |'trapNumerics|) (val))
    (declare (special |$BreakMode| |$numericFailure| |$oldBreakMode|))
    (setq val (catch |'trapNumerics| ,form))
    (if (eq val |$numericFailure|) val (cons 0 val))))

38.25   SingleInteger

defun qsquotient

--- defun qsquotient 0 ---

(defun qsquotient (a b)
  (the fixnum (truncate (the fixnum a) (the fixnum b)))))

---

defun qsremainder

--- defun qsremainder 0 ---

(defun qsremainder (a b)
  (the fixnum (rem (the fixnum a) (the fixnum b)))))

---

defmacro qsdifference

--- defmacro qsdifference 0 ---

(defmacro qsdifference (x y)
  '(the fixnum (- (the fixnum ,x) (the fixnum ,y))))

---
defmacro qslessp

This is also used in IndexedString

(defmacro qslessp (a b)
  '(< (the fixnum ,a) (the fixnum ,b)))

---

defmacro qsadd1

(defmacro qsadd1 (x)
  '(the fixnum (1+ (the fixnum ,x))))

---

defmacro qssub1

(defmacro qssub1 (x)
  '(the fixnum (1- (the fixnum ,x))))

---

defmacro qsminus

(defmacro qsminus (x)
  '(the fixnum (minus (the fixnum ,x))))
defmacro qplus

— defmacro qplus 0 —

(defmacro qplus (x y)
  `(the fixnum (+ (the fixnum ,x) (the fixnum ,y))))

------

defmacro qtimes

— defmacro qtimes 0 —

(defmacro qtimes (x y)
  `(the fixnum (* (the fixnum ,x) (the fixnum ,y))))

------

defmacro qabsval

— defmacro qabsval 0 —

(defmacro qabsval (x)
  `(the fixnum (abs (the fixnum ,x))))

------

defmacro qoddp

— defmacro qoddp 0 —

(defmacro qoddp (x)
  `(oddp (the fixnum ,x)))

------
defmacro qszerop

— defmacro qszerop 0 —

(defmacro qszerop (x)
  `(zerop (the fixnum ,x)))

defmacro qsmax

— defmacro qsmax 0 —

(defmacro qsmax (x y)
  `(the fixnum (max (the fixnum ,x) (the fixnum ,y))))

defmacro qsmin

— defmacro qsmin 0 —

(defmacro qsmin (x y)
  `(the fixnum (min (the fixnum ,x) (the fixnum ,y))))

38.26 Table

defun Table InnerTable support

We look inside the Key domain given to Table and find if there is an equality predicate associated with the domain. If found then Table will use a HashTable representation, otherwise it will use an AssociationList representation.

[knownEqualPred p??]
[compiledLookup p1185]
[Boolean p641]
[bpiname p??]
(defun hashable (dom)
  (labels ((knownEqualPred (dom)
      (let ((fun (compiledLookup '=(|Boolean|) $ $) dom)))
      (if fun
        (get (bpiname (car fun)) '|SPADreplace!)
        nil)))
    (member (knownEqualPred dom) '(eq eql equal))))

38.27 U8Vector

defmacro qvlenU8

(defmacro qvlenU8 (v)
  `(length (the (simple-array (unsigned-byte 8) (*)) ,v)))

---

defmacro eltU8

(defmacro eltU8 (v i)
  `(aref (the (simple-array (unsigned-byte 8) (*)) ,v) ,i))

---

defmacro seteltU8

(defmacro seteltU8)
(defmacro seteltU8 (v i s)
  `(setf (aref (the (simple-array (unsigned-byte 8) (*)) ,v) ,i), s))

defun getRefvU8

  — defun getRefvU8 —

(defun getRefvU8 (n x)
  (make-array n :initial-element x :element-type '(unsigned-byte 8)))

38.28 U16Vector

defmacro qvlenU16

  — defmacro qvlenU16 —

(defmacro qvlenU16 (v)
  `(length (the (simple-array (unsigned-byte 16) (*)) ,v)))

defmacro eltU16

  — defmacro eltU16 —

(defmacro eltU16 (v i)
  `(aref (the (simple-array (unsigned-byte 16) (*)) ,v) ,i))

defmacro seteltU16

  — defmacro seteltU16 —
(defmacro seteltU16 (v i s)
  '(setf (aref (the (simple-array (unsigned-byte 16) (*)) ,v) ,i), s))

defun getRefvU16

  — defun getRefvU16 —

(defun getRefvU16 (n x)
  (make-array n :initial-element x :element-type '(unsigned-byte 16)))

38.29 U32Vector

defmacro qvlenU32

  — defmacro qvlenU32 —

(defmacro qvlenU32 (v)
  '(length (the (simple-array (unsigned-byte 32) (*)) ,v)))

defmacro eltU32

  — defmacro eltU32 —

(defmacro eltU32 (v i)
  '(aref (the (simple-array (unsigned-byte 32) (*)) ,v) ,i))

defmacro seteltU32

  — defmacro seteltU32 —
(defmacro seteltU32 (v i s)
  '(setf (aref (the (simple-array (unsigned-byte 32) (*)) ,v) ,i), s))

---

defun getRefvU32

---

(defun getRefvU32 (n x)
  (make-array n :initial-element x :element-type '(unsigned-byte 32)))

---

38.30 U8Matrix

defmacro aref2U8

---

(defun aref2U8 (v i j)
  '(aref (the (simple-array (unsigned-byte 8) (* *)) ,v) ,i ,j))

---

defmacro setAref2U8

---

(defun setAref2U8 (v i j s)
  '(setf (aref (the (simple-array (unsigned-byte 8) (* *)) ,v) ,i ,j), s))

---

defmacro anrowsU8

---
(defmacro anrowsU8 (v)
  '(array-dimension (the (simple-array (unsigned-byte 8) (* *)) ,v) 0))

(defmacro ancolsU8
  (defmacro ancolsU8 —)

(defmacro ancolsU8 (v)
  '(array-dimension (the (simple-array (unsigned-byte 8) (* *)) ,v) 1))

(defmacro makeMatrixU8
  (defmacro makeMatrixU8 —)

(defmacro makeMatrixU8 (n m)
  '(make-array (list ,n ,m) :element-type '(unsigned-byte 8)
                :initial-element 0))

(defmacro makeMatrix1U8
  (defmacro makeMatrix1U8 —)

(defmacro makeMatrix1U8 (n m s)
  '(make-array (list ,n ,m) :element-type '(unsigned-byte 8)
                :initial-element ,s))

38.31  U16Matrix

defmacro aref2U16
(defmacro aref2U16 (v i j)
  ' (aref (the (simple-array (unsigned-byte 16) (* *)) ,v) ,i ,j))

(defmacro setAref2U16 (v i j s)
  '(setf (aref (the (simple-array (unsigned-byte 16) (* *)) ,v) ,i ,j) ,s))

(defmacro anrowsU16 (v)
  '(array-dimension (the (simple-array (unsigned-byte 16) (* *)) ,v) 0))

(defmacro ancolsU16 (v)
  '(array-dimension (the (simple-array (unsigned-byte 16) (* *)) ,v) 1))

(defmacro makeMatrixU16 (v)
  '(let ((x (make-array (anrowsU16 v) (ancolsU16 v))))
    (setf x))
(defmacro makeMatrixU16 (n m)
  `(make-array (list ,n ,m) :element-type '(unsigned-byte 16)
                :initial-element 0))

---

defmacro makeMatrix1U16

---

defmacro aref2U32

---

defmacro setAref2U32

---
defmacro anrowsU32

— defmacro anrowsU32 —

(defun anrowsU32 (v)
  `(array-dimension (the (simple-array (unsigned-byte 32) (* *)) ,v) 0))

defmacro ancolsU32

— defmacro ancolsU32 —

(defun ancolsU32 (v)
  `(array-dimension (the (simple-array (unsigned-byte 32) (* *)) ,v) 1))

defmacro makeMatrixU32

— defmacro makeMatrixU32 —

(defun makeMatrixU32 (n m)
  `(make-array (list ,n ,m) :element-type '(unsigned-byte 32)
               :initial-element 0))

defmacro makeMatrix1U32

— defmacro makeMatrix1U32 —

(defun makeMatrix1U32 (n m s)
  `(make-array (list ,n ,m) :element-type '(unsigned-byte 32)
               :initial-element ,s))
38.33 U32VectorPolynomialOperations

defmacro qsMulAdd6432

   — defmacro qsMulAdd6432 —

(defmacro qsMulAdd6432 (x y z)
   '(the (unsigned-byte 64)
      (+ (the (unsigned-byte 64)
        (* (the (unsigned-byte 32) ,x)
        (the (unsigned-byte 32) ,y)))
      (the (unsigned-byte 64) ,z))))

———

defmacro qsMulMod32

   — defmacro qsMulMod32 —

(defmacro qsMulMod32 (x y)
   '(the (unsigned-byte 64)
      (* (the (unsigned-byte 32) ,x)
      (the (unsigned-byte 32) ,y))))

———

defmacro qsMod6432

   — defmacro qsMod6432 —

(defmacro qsMod6432 (x p)
   '(the (unsigned-byte 32)
      (rem (the (unsigned-byte 64) ,x) (the (unsigned-byte 32) ,p))))

———

defmacro qsMulAddMod6432

   — defmacro qsMulAddMod6432 —
(defmacro qsMulAddMod6432 (x y z p)
  '(qsMod6432 (qsMulAdd6432 ,x ,y ,z) ,p))

---

defmacro qsMul6432

— defmacro qsMul6432 —

(defmacro qsMul6432 (x y)
  '(the (unsigned-byte 64)
    (* (the (unsigned-byte 32) ,x)
      (the (unsigned-byte 32) ,y))))

---

defmacro qsDot26432

— defmacro qsDot26432 —

(defmacro qsDot26432 (a1 b1 a2 b2)
  '(qsMulAdd6432 ,a1 ,b1 (qsMul6432 ,a2 ,b2)))

---

defmacro qsDot2Mod6432

— defmacro qsDot2Mod6432 —

(defmacro qsDot2Mod6432 (a1 b1 a2 b2 p)
  '(qsMod6432 (qsDot26432 ,a1 ,b1 ,a2 ,b2) ,p))

---

38.34 Void
defun voidValue
38.34. VOID

— defun voidValue —

(defun voidValue () "()")
CHAPTER 38. COMMON LISP ALGEBRA SUPPORT
Chapter 39

OpenMath

39.1 A Technical Overview

OpenMath\[Dewa\] is a standard for representing mathematical data in as unambiguous a way as possible. It can be used to exchange mathematical objects between software packages or via email, or as a persistent data format in a database. It is tightly focussed on representing semantic information and is not intended to be used directly for presentation, although tools exist to facilitate this.

The original motivation for OpenMath came from the Computer Algebra community. Computer Algebra packages were getting bigger and more unwieldy, and it seemed reasonable to adopt a generic "plug and play" architecture to allow specialised programs to be used from general purpose environments. There were plenty of mechanisms for connecting software components together, but no common format for representing the underlying data objects. It quickly became clear that any standard had to be vendor-neutral and that objects encoded in OpenMath should not be too verbose. This has led to the design outlined below.

In 1998, the Worldwide Web Consortium (W3C) produced its first recommendation for the Extensible Markup Language (XML), intended to be a universal format for representing structured information on the worldwide web. It was swiftly followed by the first MathML recommendation which is an XML application oriented mainly towards the presentation (i.e. the rendering) of mathematical expressions.

The formal definition of OpenMath is contained within The OpenMath Standard and its accompanying documents, and the reader is referred there for more details.

The OpenMath Architecture

The OpenMath representation of a mathematical structure is referred to as an OpenMath object. This is an abstract structure which is represented concretely via an OpenMath encoding. These encoded objects are what an OpenMath application would read and write,
and in practice the OpenMath objects themselves almost never exist, except on paper. The advantage of this is that OpenMath is not tied to any one underlying mechanism: in the past we have used functional, SGML and binary encodings. The current favourite is XML, as described below, and we will tend to use XML notation when describing OpenMath objects (even though strictly speaking the XML representation is an encoding). OpenMath Objects Formally, an OpenMath object is a labelled tree whose leaves are the basic OpenMath objects integers, IEEE double precision floats, unicode strings, byte arrays, variables or symbols. Of these, symbols are the most interesting since they consist of a name and a reference to a definition in an external document called a content dictionary (or CD). Using XML notation where the element name OMS indicates an OpenMath symbol, the following:

\[
\text{<OMS name="sin" cd="transc1"/>}
\]

represents the usual sine function, as defined in the CD "transc1". A basic OpenMath object is an OpenMath object, although its XML representation will be:

\[
\text{<OMOBJ>}
\text{<OMS name="sin" cd="transc1"/>}
\text{</OMOBJ>}
\]

OpenMath objects can be built up recursively in a number of ways. The simplest is function application, for example the expression \(\sin(x)\) can be represented by the XML:

\[
\text{<OMOBJ>}
\text{<OMA>}
\text{<OMS name="sin" cd="transc1"/>}
\text{<OMV name="x"/>}
\text{</OMA>}
\text{</OMOBJ>}
\]

where OMV introduces a variable and OMA is the application element. Another straightforward method is attribution which as the name suggests can be used to add additional information (for example "the AXIOM command which generated me was ...") to an object without altering its fundamental meaning. More interesting are binding objects which are used to represent an expression containing bound variables, for example:

\[
\text{<OMOBJ>}
\text{<OMA>}
\text{<OMS cd="calculus1" name="int"/>}
\text{<OMS cd="transc1" name="sin"/>}
\text{</OMA>}
\text{</OMOBJ>}
\]

represents the integral of the sin function, but the encoding:

\[
\text{<OMOBJ>}
\text{<OMA>}
\text{<OMS cd="calculus1" name="int"/>}
\text{<OMS cd="transc1" name="sin"/>}
\text{</OMA>}
\text{</OMOBJ>}
\]
represents \( \sin(x)dx \). This may appear overly complicated but it is useful, for example when searching in a database for expressions which match \( \sin(y)dy \). The definition of a symbol in the CD specifies whether or not it may be used to bind variables, which is why

\[
<\text{OMS cd="calculus1" name="int"} />
<\text{OMBIND}>
<\text{OMS cd="fns1" name="lambda"} />
<\text{OMBVAR} <\text{OMV name="x"} /> <\text{OMBVAR}>
<\text{OMA}>
<\text{OMS name="sin" cd="transc1"} />
<\text{OMV name="x"} />
</\text{OMA}>
</\text{OMBIND}>
</\text{OMA}>
</\text{OMOBJ}>
\]

cannot be used as a binding symbol.

The final kind of OpenMath object is an error which is built up from a symbol describing the error and a sequence of OpenMath objects. For example:

\[
<\text{OMOBJ}>
<\text{OME}>
<\text{OMS name="unexpected_symbol" cd="error1"} />
<\text{OMS name="sine" cd="transc1"} />
</\text{OME}>
</\text{OMOBJ}>
\]

represents the error which might be generated when an application sees a symbol it doesn’t recognise from a CD it thought it knew about.

**OpenMath Encodings**

We have already seen some examples of the XML encoding, but it is by no means the only encoding. In the past there was a functional encoding (which looked like Lisp) and an SGML encoding which evolved into the current XML. Both of these are now obsolete, but there is still a binary encoding described in the standard, which is much more compact than the XML one.

In fact the XML encoding is not 100% XML. When XML was in its infancy the developers of OpenMath realised that it might become significant and decided to add some XML-like features to the SGML encoding so that an an OpenMath object could be encoded as valid XML. Thus it is currently the case that any well-formed OpenMath object encoded using the XML encoding as described in the standard is a valid XML document. However, if one uses standard XML tools to generate an OpenMath object in the XML encoding from the DTD given in chapter 4 of the standard, it is possible that the result will not be valid OpenMath,
although in practice this is highly unlikely. To cover all the possibilities allowed by XML would make it much more complicated to write an application to read any OpenMath object from scratch. Whether to adopt XML completely remains a hot topic of debate within the OpenMath community!

Generally speaking, it is not intended that the existing encodings should be readable by a human user or writable by hand. It is desirable that they be compact and it is also desirable that they be linear, but neither of these is a requirement. It is a property of encodings that it is possible to convert between them with no loss of information.

**Content Dictionaries**

Content Dictionaries (or CDs for short) are the most important, and the most interesting, aspect of OpenMath because they define the meaning of the objects being transmitted. A CD is a collection of related symbols and their definitions, encoded in an XML format. Defining the meaning of a symbol is not a trivial task, and even referring to well-known references can be fraught with pitfalls Formal definitions and properties can be very useful but time-consuming to produce and verbose, not to mention difficult to get right. A symbol definition in an OpenMath CD consists of the following pieces of information:

- the symbol name;
- a description in plain text;
- optionally, a set of this symbol’s properties in plain text (Commented Mathematical Properties, or CMPs);
- optionally, a set of this symbol’s properties encoded in OpenMath (Formal Mathematical Properties, or FMPs);
- optionally, one or more examples of its use (encoded in OpenMath).

In practice the CMPs and FMPs can come as pairs, and often serve in the place of examples. A very simple instance of a CD definition is:

```xml
<CDDefinition>
  <Name> log </Name>

  <Description>
  This symbol represents a binary log function; the first argument is the base, to which the second argument is log'ed.
  It is defined in Abramowitz and Stegun, Handbook of Mathematical Functions, section 4.1
  </Description>

  <CMP>
  a \times b = c \implies \log_a c = b
  </CMP>

  <FMP>
  <OMOBJ>
    <OMA>
      <OMS cd="logic1" name="implies"/>
    </OMA>
  </OMOBJ>
  </FMP>
</CDDefinition>
```
Another example would be to print the list

\[ 1, \frac{1}{2} \]

as

\[ \text{OMOBJ} \]
\[ \text{OMA} \]
\[ \text{OMS cd="list1" name="list"} \]
\[ \text{OMI}1\text{/OMI} \]
\[ \text{OMA} \]
\[ \text{OMS cd="nums1" name="rational"} \]
\[ \text{OMI}1\text{/OMI} \]
This provides a symbol to represent the log function by giving a pointer to a standard reference book. It provides the property that:

$$a^b = c \rightarrow \log_a(c) = b$$

both as plain text and as OpenMath, and also gives an example of how the symbol is used. CDs usually consist of related symbols and collections of related CDs can be grouped together, for convenience, as CD Groups. One very important CD Group is that corresponding to the content part of MathML.

It is possible to associate extra information with CDs, in particular type information. Since there are many type systems available, each of which has its own strengths and advocates, the OpenMath community does not mandate any single system. Simple signatures can be encoded using the Simple Type System, while more formal definitions are possible using the Extended Calculus of Constructorss. Other associated information can include style sheets for rendering OpenMath symbols in MathML, and mathematical definitions to be used by formal logic systems.

Given the evolutionary nature of mathematics, it is clear that the set of CDs should be forever growing and never complete. Currently there are CDs for high-school mathematics, linear algebra, polynomials and group theory to name a few, and new contributions are always welcome. There is no requirement that applications use the standard set of CDs and it is often very useful to design a "private" CD for a specific purpose.

**OpenMath in Action**

There is no definitive way in which OpenMath should be used, as the protocol has been designed to be as flexible as possible. Nevertheless many OpenMath applications share common characteristics which we shall discuss here.

Suppose that we wish to have two applications communicating by sending OpenMath objects to each other, e.g. a client program and a computational server. It is unlikely that the internal data structures used by the applications will be OpenMath, and so translation between the internal representations and OpenMath (almost certainly OpenMath encodings rather than objects) will have to take place. The piece of software which does this is usually referred to as a phrase-book.

It is possible to write a generic phrase-book which can handle any piece of OpenMath, but applications where this makes sense are few and far between. In practice an OpenMath phrase book will usually only handle a fixed set of CDs (and hence a fixed set of symbols). What "handle" means will vary from case to case: a computer algebra system will usually try and evaluate its input and return a result or an error, while a typesetter will print its input according to some rendering rules and not return anything. OpenMath carefully avoids
defining what the “right” behaviour is in a given circumstance, and leaves that up to the phrase-book writer. Indeed it is quite possible that a piece of software could have multiple phrase-books associated with it for different purposes. OpenMath symbols should not be regarded as verbs since they are used to construct objects rather than to send commands, and the presence of both nouns and verbs in a CD (e.g. “integral” and “integrate”) is strongly discouraged.

Writing a phrase-book may be non-trivial, and requires an understanding of the semantics of the underlying software. An OpenMath object may not map directly into a private object and vice-versa, for example in some systems a rational number might have to be represented by a float, or a sparse matrix by a dense one.

The OpenMath standard includes a section on compliance, which describes the behaviour of an OpenMath application when certain errors occur. It also insists that all compliant software has the capability to use the XML encoding, to guarantee a degree of interoperability. This is an area where the standard is expected to evolve as more OpenMath applications become available.

39.2 Technical Details

This chapter describes the Axiom implementation of the OpenMath project [Dalm97] at INRIA. The code enables the exchange of OpenMath objects between two processes and more generally the input and output of OpenMath objects. First we describe the library API and then we implement the functions used by Axiom.

39.3 The Structure of the API

The library and its API are logically structured in four parts:

- Functions that deal with devices, the abstraction from which OpenMath objects are read and written to.
- Functions that read from and write to OpenMath devices. These functions use a simple model that read and write tokens.
- Functions that create I/O structures to be used by devices, so that, for example, an OpenMath object can be read from a file or a socket. This part is extensible by the user.
- Functions that deal with interprocess communication.
39.4 OpenMath Expressions

Expressions

The library understands the following kinds of basic OpenMath expressions:

- integers
- double precision floating-point numbers (64 bits, following IEEE 754)
- byte arrays
- character strings
- symbols
- variables

and the four kinds of constructions:

- applications $e_0(e_1, \ldots, e_n)$
- errors $s(e_1, \ldots, e_n)$
- binders $e_1, (v_1, \ldots, v_n), e_2$
- attributed expressions $[s_1 e_1, \ldots, s_n e_n] e$

where $e_i$ are OpenMath expressions, $v_i$ are OpenMath variables and $s$ and $s_i$ are OpenMath symbols.

Symbols

Symbols are constructed from a content dictionary (abbreviated as CD in the sequel) and a name. A content dictionary is identified by its name. The API permits the creation of any symbol in any content dictionary: there is nothing that prevents creating symbols that do not belong to a known CD.

Encoding and Decoding OpenMath Expressions

An OpenMath object is encoded as a sequence of bytes that is read and written sequentially. The library views this sequence as a stream of tokens. Expressions are linearized in a way that looks like Lisp with typed parenthesis. For example, the linearization of the application of $S$ to $E_1 \ldots E_n$ is:

- indicating that this is an application (a “begin application” token)
- linearizing $S$
• linearizing $E_1 \ldots E_n$
• indicating that all arguments have been given (an “end application” token)

The other constructions are linearized the same way (each one with its own begin and end tokens). Note that there is no explicit arity indication so that we don’t have to introduce a special mechanism when we don’t know beforehand how many arguments there are.

To give attributes to an expression, the attributes and their associated values are put before the expression. To give the attributes $a_i$ with values $v_i$ (where $a_i$ are symbols and $v_i$ are OpenMath expressions) to an expression $E$ the process is:

• put a “begin attributed expression” token
• put a “begin attribute pairs” token
• put the symbol $a_1$ followed by the linearization of $v_1$ etc
• put an “end attribute pairs” token
• linearize $E$
• put an “end attributed expression” token

Decoding is done by first querying the type of the next OpenMath token and then invoking the right function to get this particular kind of token.

### 39.5 Big Integers

The library supports big integers that can potentially be given in various formats. The `OMBigIntType` describes the different possible formats.

```c
typedef enum OMbigIntType {
    OMBlUnknown = 0, /* this is base 10, digits in normal order */
    OMBlBase10,    /* this is base 16, digits in normal order (MSB) */
    OMBlBase16
} OMbigIntType;
```

### 39.6 Functions Dealing with OpenMath Devices

OpenMath expressions are read and written through devices. Basically, an OpenMath device has an associated encoding and an I/O method. There are basically two encodings defined and implemented. The first one is a human readable and writable one that can be used for example as the encoding for sending OpenMath objects via e-mail or storing OpenMath objects to files. This encoding is SGML compatible in the sense that it can be used to represent OpenMath objects in SGML texts. It has an XML variant. The second encoding
is a binary one that can be used when compactness and speed of encoding and decoding is important. The encodings are defined by the \texttt{OMencodingType} type which is an enumerated type defined as

\begin{verbatim}
typedef enum OMencodingType
{
    OMencodingUnknown,
    OMencodingBinary,
    OMencodingSGML,
    OMencodingXML} OMencodingType;
\end{verbatim}

\texttt{OMencodingUnknown} is to be used when creating a device that does not know which kind of encoding will be used. It must be used only for input devices.

A device is created with the following function, given an encoding and an appropriate I/O method:

\begin{itemize}
  \item \texttt{OMdev OMmakeDevice(OMencodingType encoding, OMIO IO)}
\end{itemize}

Devices are closed with the following function

\begin{itemize}
  \item \texttt{void OMcloseDevice(OMdev dev)}
\end{itemize}

Whether a device could be used both for reading and writing is entirely dependent on its I/O method.

The user can define its own I/O method as a function returning an \texttt{OMIO} object. This could enable him, for example, to use an existing transport protocol to exchange OpenMath expressions or to implement cut-and-paste of OpenMath expression by writing I/O structures that input and output to strings. The I/O section describes the available I/O structures in the library.

An \texttt{OMdev} object is a pointer to a structure that contains a lot of state. Almost all functions taking an \texttt{OMdev} object modify it. Likewise, an \texttt{OMIO} object carries a lot of state.

### 39.7 Functions to Write OpenMath Expressions to Devices

#### Beginning and Ending Objects

The following two functions mark the beginning and end of an OpenMath object.

\begin{itemize}
  \item \texttt{OMstatus OMputObject(OMdev dev)}
  \item \texttt{OMstatus OMputEndObject(OMdev dev)}
\end{itemize}

These functions should be called before and after an OpenMath object in constructed in a device. In particular, the \texttt{OMputEndObject} function insures that the object has been completely written if any buffering was used.
Writing Basic Objects

Basic OpenMath objects are written using these functions:

- `OMstatus OMputInt32(OMdev dev, int n)`
- `OMstatus OMputBigInt(OMdev dev, const char *data, int len, int sign, OMbigIntType format)`
- `OMstatus OMputFloat64(OMdev dev, double *f)`
- `OMstatus OMputByteArray(OMdev dev, const char *data, int len)`
- `OMstatus OMputString(OMdev dev, const char *s)`
- `OMstatus OMputVar(OMdev dev, const char *name)`
- `OMstatus OMputSymbol(OMdev dev, const char *cd, const char *name)`

The `char *` arguments of `OMputString`, `OMputVar` and `OMputSymbol` are null-terminated strings. There are other functions that accept non null-terminated arrays of characters with their length. These are

- `OMstatus OMputStringN(OMdev dev, const char *str, int len)`
- `OMstatus OMputVarN(OMdev dev, const char *var, int len)`
- `OMstatus OMputSymbolN(OMdev dev, const char *cd, int clen, const char *name, int nlen)`

The format for the data argument of the `OMputBigInt` function is given by `format`. When `format` is `OMBIbase10`, it is the sequence of character of its base 10 representation without sign (most significant digit first). The sign of the big integer is given by the `sign` argument that should be an integer greater or equal to zero for a positive integer and less than zero for a negative one. For example, the following line outputs the value of 20! to `dev`:

```
OMputBigInt(dev, "265252859812191058636308480000000", 33, 1, OMBIbase10);
```

Writing Structured Objects

The following functions are used to mark the beginning and end of the structured objects. They should be called in nested pairs, correctly bracketed:

- `OMstatus OMputApp(OMdev dev)`
- `OMstatus OMputEndApp(OMdev dev)`
- `OMstatus OMputAttr(OMdev dev)`
- `OMstatus OMputEndAttr(OMdev dev)`
Here is an example showing how to use these functions to output \( \sin x + y \), where \( x \) and \( y \) are represented as variables and \( \sin \) is the symbol whose name is \texttt{sin} in the \texttt{Basic} content dictionary. This can be done using the following sequence:

```c
OMputObject(dev);
OMputApp(dev);
    OMputSymbol(dev, "Basic", "sin");
    OMputApp(dev)
        OMputSymbol(dev, "Basic", "+");
        OMputVar(dev, "x");
        OMputVar(dev, "y");
    OMputEndApp(dev);
OMputEndApp(dev);
OMputEndObject(dev);
```

### 39.8 Functions to Extract OpenMath Expressions from Devices

#### Testing the type of the current token

The first step in decoding an expression from a device is to call the \texttt{OMgetType} function

```c
OMstatus OMgetType(OMdev dev, OMtokenType *type)
```

so that the correct function can be called to recover the current token. \texttt{OMgetType} returns via its \texttt{type} argument an \texttt{OMtokenType} object indicating the type of the next object to be read from the device. \texttt{OMtokenType} is an enumerated type defined as

```c
typedef enum OMtokenType {
    OMtokenUnknown, /* error catching trick */
    OMtokenInt32,
```
39.8. FUNCTIONS TO EXTRACT OPENMATH EXPRESSIONS FROM DEVICES

OMtokenBigInt,
OMtokenFloat64,
OMtokenByteArray,
OMtokenVar,
OMtokenString,
OMtokenSymbol,
OMtokenComment,
OMtokenApp, OMtokenEndApp,
OMtokenAttr, OMtokenEndAttr,
OMtokenAtp, OMtokenEndAtp,
OMtokenError, OMtokenEndError,
OMtokenObject, OMtokenEndObject,
OMtokenBind, OMtokenEndBind,
OMtokenBVar, OMtokenEndBVar,
} OMtokenType;

Note that the type of the current token can be tested multiple times. Two successive calls to OMgetType will always return the same result if no other OMget... function was called in between.

Extracting the current token

The following functions are used to read the basic OpenMath objects from devices:

- OMstatus OMgetInt32(OMdev dev, int *i)
- OMstatus OMgetFloat64(OMdev dev, double *d)
- OMstatus OMgetBigInt(OMdev dev, char **data, int *len, int *sign, OMbigIntType *fmt)
- OMstatus OMgetBigIntN(OMdev dev, char *data, int len, int *sign, OMbigIntType *fmt)
- OMstatus OMgetByteArray(OMdev dev, char **data, int *len)
- OMstatus OMgetByteArrayN(OMdev dev, char *data, int len)
- OMstatus OMgetString(OMdev dev, char **str)
- OMstatus OMgetStringN(OMdev dev, char *str, int len)
- OMstatus OMgetVar(OMdev dev, char **var)
- OMstatus OMgetVarN(OMdev dev, char *var, int len)
- OMstatus OMgetSymbol(OMdev dev, char **cd, char **name)
- OMstatus OMgetSymbolN(OMdev, char *cd, int clen, char *name, int nlen)
The functions that return variable size data exist in two versions. A simple version that does the necessary memory allocation itself (using `OMmalloc`) and a version (suffixed with `N`) that lets the user do the allocation itself. The size of the needed area can be determined with the following function:

- `int OMgetLength(OMdev dev)` returns the length of the next object.

That works for big integers, byte arrays, strings and variables. For symbols, the following function returns both the length of the content dictionary name and the length of the symbol name:

- `OMstatus OMgetSymbolLength(OMdev dev, int *clen, int *nlen)`

When the current token does not carry any data i.e. when `OMgetType` returns a marker, i.e. one of:

- `OMtokenApp`,
- `OMtokenEndApp`,
- `OMtokenAttr`,
- `OMtokenEndAttr`,
- `OMtokenAtp`,
- `OMtokenEndAtp`,
- `OMtokenError`,
- `OMtokenEndError`,
- `OMtokenObject`,
- `OMtokenEndObject`,
- `OMtokenBind`,
- `OMtokenEndBind`,
- `OMtokenBVar`
- `OMtokenEndBVar`

It is necessary to call the correct function to remove the marker. The available functions are:

- `OMstatus OMgetObject(OMdev dev)`
- `OMstatus OMgetEndObject(OMdev dev)`
- `OMstatus OMgetApp(OMdev dev)`
39.8. FUNCTIONS TO EXTRACT OPENMATH EXPRESSIONS FROM DEVICES

- OMstatus OMgetEndApp(OMdev dev)
- OMstatus OMgetAttr(OMdev dev)
- OMstatus OMgetEndAttr(OMdev dev)
- OMstatus OMgetAtp(OMdev dev)
- OMstatus OMgetEndAtp(OMdev dev)
- OMstatus OMgetBind(OMdev dev)
- OMstatus OMgetEndBind(OMdev dev)
- OMstatus OMgetBVar(OMdev dev)
- OMstatus OMgetEndBVar(OMdev dev)
- OMstatus OMgetError(OMdev dev)
- OMstatus OMgetEndError(OMdev dev)

All the previous functions return OMsuccess when they succeed. When they return something else, there has been a problem such as calling the wrong function (OMgetApp when there is not a “beginning of application” mark) or a system error.

The sequence of calls to read an expression is thus completely similar (if we omit the calls to OMgetType) to the sequence of calls to write the expression. For example, the previous expression \((\sin x + y)\) can be recovered via the sequence:

```c
OMgetObject(dev);
OMgetApp(dev);
    OMgetSymbol(dev, ...);
    OMgetApp(dev);
    OMgetSymbol(dev, ...);
    OMgetVar(dev, ...);
    OMgetVar(dev, ...);
OMgetEndApp(dev);
OMgetEndApp(dev);
OMgetEndObject(dev);
```

OMgetInt32(OMdev dev, int *i) returns the integer through its i argument.

OMgetBigInt(OMdev dev, char **data, int *len, int *sign, OMbigIntType *fmt)

returns the data corresponding to the big integer in data, its length in len, its sign in sign and its format in fmt.

OMgetBigIntN(OMdev dev, char *data, int len, int *sign, OMbigIntType *fmt)
copies the data corresponding to the big integer in data buffer that should be (at least) len characters long. The sign and format are returned in the sign and fmt arguments.

OMgetByteArray(OMdev dev, char **data, int *len) returns the byte array through its data argument. Its length is returned via the len argument.

OMgetByteArrayN(OMdev dev, char *data, int len) copies the byte array in the data buffer that should be (at least) len characters long.

OMgetString(OMdev dev, char **str) returns the string through its str argument.

OMgetStringN(OMdev dev, char *str, int len) copies the string in the str buffer whose length should be (at least) len. If len is greater than the actual length of the string, a null character is added at the end of str.

OMgetVar(OMdev dev, char **var) returns the name of the variable (as a null-terminated string) in its var argument.

OMgetVarN(OMdev dev, char *var, int len) copies the name of the variable in the var buffer, whose length should be (at least) len. If len is greater than the actual length of the variable name, a null character is added at the end of var.

OMgetSymbol(OMdev dev, char **cd, char **name) returns the content dictionary and the name of the symbol through the cd and name arguments.

OMgetSymbolN(OMdev dev, char *cd, int clen, char *name, int nlen) copies the content dictionary and the name of the symbols in the cd and name buffers. cd should be at least clen character long and name should be at least nlen long. When there is enough room (based on clen or nlen) a null character is added after the last character of the name (cd or name).

### 39.9 Comments in the SGML/XML Encodings

The library can also output and read comments (SGML/XML comments) with the following functions:

- OMstatus OMputComment(OMdev dev, char *comment)
- OMstatus OMputCommentN(OMdev dev, char *comment, int len)
- OMstatus OMgetComment(OMdev dev, char **comment)
- OMstatus OMgetCommentN(OMdev dev, char *comment, int len)

By default, comments are silently ignored by the library when reading OpenMath objects (and writing them using the binary encoding). The function

- OMbool OMignoreComment(OMdev dev, OMbool set)

changes this behaviour. When called with OMfalse, comments are passed to the application: the OMgetType function will return OMtokenComment when the current token is a comment and the OMgetComment or OMgetCommentN functions should be used to get the comments. When OMignoreComment is called with OMtrue, comments are ignored.
39.10 I/O Functions for Devices

We provide four functions that produce OMIO objects for devices. These functions provide I/O through the *stdio* library (on FILE object), file descriptors and character strings.

- OMIO OMmakeIOFile(FILE *f) associates the device with the file pointer f.
- OMIO OMmakeIOfd(int fd) associates the device with the file descriptor fd.
- OMIO OMmakeIOHandle(HANDLE handle) associates the device with a file handle *Windows specific version of OMmakeIOfd().fd.*
- OMIO OMmakeIOString(char **s) associates the device with a string.

For example, the following code opens a device that reads from standard input:

```c
dev = OMmakeDevice(OMencodingSGML, OMmakeIOFile(stdin));
```

The OMmakeIOString builds an input device that reads from a string or an output device that writes to a string. For input, s must point to a character string (null terminated). For output, s will point to a string allocated by the library (note that the string s points to can be reallocated by the library).

39.11 Communications

A communication layer can be put above the device layer. In fact, the I/O structure in a device provides all the necessary support to use any transmission or communication means. This library directly provides some connection-oriented, client-server facilities (based on TCP).

A set of functions are used to set up connections. Connections are described by the OMconn type. An OMconn is a (pointer to a) structure with two user-accessible fields in and out. in is a pointer to a device to be used for input. out is pointer to a device to be used for output. These devices use the binary encoding.

An OMconn object is made with the following function:

- OMconn OMmakeConn(int timeout)

where timeout is a timeout for the connection, expressed in milliseconds.

- OMdev OMconnIn(OMconn conn) returns the input device associated with the connection.
- OMdev OMconnOut(OMconn conn) returns the output device associated with the connection.
Functions to Initiate an OMconn

The functions we provide can be divided in two classes. The first one simply establishes an interprocess communication using IP addresses. The second one provides functions that can be used to launch a server. The addresses used are then generated by the library.

Simple Connections Functions

The following functions allow a client OpenMath application to contact an OpenMath server at a specified address:

- OMstatus OMconnTCP(OMconn conn, char *machine, int port)
- OMstatus OMconnUnix(OMconn conn, char *file)

These functions first physically establish the connection. Then, they enter negotiation with the server (they send the first message). When they return, the negotiation is finished and the devices in the conn argument are ready.

On the server side, the following functions provide bindings at specified addresses and take care of the negotiation:

- OMstatus OMbindTCP(OMconn conn, int port)
- OMstatus OMbindUnix(OMconn conn, char *file)

All four the previous functions block until the connection is established (and negotiation is over) or the timeout of the conn argument is reached.

The following function returns the file descriptor associated with a device. This is intended to be used when there is a need to poll the device (through the select or poll system calls).

- OMdeviceFd(OMdev dev)

Functions that Launch Servers

These functions provide the same functionalities for launching a server that were provided in the ASAP library.

In this model, the client calls OMLaunch with a machine name mach and a string cmd that is executed via rsh on machine mach as a shell command line. This command is supposed to launch the server program. The command is executed in an environment (in the UNIX sense) where some variables are associated with an address on the machine that runs the client. The server can then connect to the client with the OMServeClient function.

If the machine name is localhost, the command is started on the same machine (without calling rsh).

- OMstatus OMLaunchEnv(OMconn conn, char *machine, char *command, char *env)
39.12 Parameters

The library internally uses three functions that can be supplied by the user.

- extern void *(OMmalloc) (size_t size)
- extern void *(OMrealloc) (void *ptr, size_t size)
- extern void *(OMfree) (void *ptr)

OMmalloc is used for all memory allocations in the library. The default value is the malloc function.

OMfree is used for deallocations. The default value is the free function.

OMfatal is invoked when a fatal error is detected in the library (for example when memory allocation failed or when an inconsistency is detected in the library code data structures). The default value just does an exit.

OMfatal is declared as extern void (*)(OMstatus status). All memory allocations and deallocations in the library are done through the OMmalloc and OMfree functions.

39.13 Miscellaneous Functions and Variables

- char *(OMstatusToString) (OMstatus status) make a status into a human readable string.
1278  CHAPTER 39. OPENMATH

- char *OMtokenTypeToString(OMtokenType ttype) makes a tokenType into a human readable string.
- OMencodingType OMgetDeviceEncoding(OMdev dev) returns the encoding actually used by the device.
- char *OMlibDynamicInfo(void)
- extern const char *OMlibVersion is the version of the library.
- extern const char *OMlibInfo contains some textual information about the library.

39.14 The OM.h header file

#ifndef __OM_h__
#define __OM_h__

/* All types used through API. */

/* These types are anonymized by the mean of a generic pointer.
* You should not allocate or dereference objects of these types.
* API (hopefully) provides you with all needed methods.
* If you find any that are not included, please refer to
* us rather than using private structures.
* i.e. If you need to do something like
*    malloc(sizeof(OMdevStruct));
* or
*    OMdevStruct *pdev;
*    pdev->anyField = something;
* this probably means we need to discuss your problem.
*/

/* A device is an abstraction for put/get of OpenMath tokens */
typedef struct OMdevStruct *OMdev;

/* IO is a device field, (the physical IO channel) */
typedef struct OMIODevice *OMIO;

/* Error status that may be returned */
typedef enum OMstatus {
    /* Last call was successful. */
    OMsuccess = 0,
    /* Last call failed for some undetermined reason. */
    OMfailed = 1,
    /* Last call failed for memory reasons. */
}
OMnoMem,
   /* Last call failed during some system call. */
OMerrorSys,
   /* Last call to some OMget* function failed due to an unexpected EOF
     on input IO. */
OMemptyIO,
   /* Last call to some OMget* function failed because there is no more
     token on device. */
OMnoMoreToken,
   /* Last call to some OMget* function timed out. */
OMtimeoutedRead,
   /* Last call to some OMget* function failed due to malformed input.
     (this error covers all low level lexical or syntactic problems). */
OMmalformedInput,
   /* Last call to OMbindTCP failed because address is already in use
     (EADDRINUSE). */
OMaddrInUse,
   /* Last call to OMconnTCP failed to set connection. */
OMconnectFailed,
   /* Last call trigered some not (yet) implemented code in this lib. */
OMnotImplemented,
   /* Last call caused some internal trouble. */
OMinternalError
} OMstatus;

/* All OpenMath token kinds are identified by one of these types.
 * Values given in this enum have been chosen to:
 * - avoid conflicts with specific XML characters
 * - to help automatic detection of encoding type.
 *   (no: '\t'(9) '\r'(13) '\n'(10) '<'(60) or ' '(32))
 * - keep some bits (3) available for special encodings purpose
 *   (eg: sharing or big len flags in binary encoding)
 */
typedef enum OMtokenType {
    OMtokenUnknown = 0, /* error catching trick */
    OMtokenInt32 = 1,
    OMtokenBigInt = 2,
    OMtokenFloat64 = 3,
    OMtokenByteArray = 4,
    OMtokenVar = 5,
    OMtokenString = 6,
    OMtokenWCString = 7,
    OMtokenSymbol = 8,
    OMtokenComment = 15,
    OMtokenApp = 16, OMtokenEndApp = 17,
    OMtokenAttr = 18, OMtokenEndAttr = 19,
    OMtokenAtp = 20, OMtokenEndAtp = 21,
    OMtokenError = 22, OMtokenEndError = 23,
    OMtokenObject = 24, OMtokenEndObject = 25,
    OMtokenBind = 26, OMtokenEndBind = 27,
}
OMtokenBVar = 28, OMtokenEndBVar = 29
} OMtokenType;

typedef enum OMbigIntType {
    OMbigIntUnknown = 0,
    /* this is base 10, digits in normal order (MSB) */
    OMbigIntBase10,
    /* this is base 16, digits in normal order (MSB) */
    OMbigIntBase16
} OMbigIntType;

/* Encodings should not be "user visible"
* We thus refer to encoding as "symbolic constants" from this enum type. */
typedef enum OMencodingType {
    /* You may set an input stream to "unknown encoding".
    * By doing this, you let library auto detect the
    * encoding type of the device during first token input. */
    OMencodingUnknown = 0,
    /* Binary encoding, more compact than XML one. */
    OMencodingBinary,
    /* XML-like encoding, human readable. */
    OMencodingXML,
} OMencodingType;

/* This is a portable equivalent to wchar_t for unicode strings */
typedef unsigned short OMUCS2;

/* Replacment for lacking C bools */
typedef unsigned char OMbool;
#define OMfalse (0)
#define OMtrue (1)

/* Some global variables */
extern const char *OMlibVersion;
extern const char *OMlibInfo;
extern void *(*OMmalloc) (size_t size);
extern void *(*OMrealloc) (void *ptr, size_t size);
extern void (*OMfree) (void *ptr);

/* If set, this function will be called by OMfatal, thus you may use it for
error handling (by default it is set to exit()) */
extern void (*OMfatal)(OMstatus status);

#ifndef OM_DEV
/* this part is automatically updated, do NOT edit below */
/** Prototypes */
/* OMPut* functions.
 * They all take a device <dev> to put token to.
 * Some of them need more parameters to define the token content.
 * They are thoroughly documented in OpenMath Specification shipped
 * with the library.
 * return: a status that reflect the operation success.
 */
extern OMstatus OMputInt32(OMdev dev, int n);
extern OMstatus OMputFloat64(OMdev dev, double *d);
extern OMstatus OMputBigInt(OMdev dev, const char *data, int len,
   int sign, OMbigIntType format);
extern OMstatus OMputByteArray(OMdev dev, const char *data, int len);
/* OMputString*
 * If you want to output plain 8bits C like strings there is no need
 * to use the OMputWCString* functions. This one is more efficient
 * (faster and more compact output for some encodings)
 */
extern OMstatus OMputString(OMdev dev, const char *str);
extern OMstatus OMputStringN(OMdev dev, const char *str, int len);
/* OMputWCString
 * If you are using wide char strings you need to output them
 * with that function rather than with OMputString.
 * (It takes endianess into account)*/
extern OMstatus OMputWCString(OMdev dev, const OMUCS2 * wcstr);
extern OMstatus OMputVar(OMdev dev, const char *var);
extern OMstatus OMputVarN(OMdev dev, const char *var, int len);
extern OMstatus OMputSymbol(OMdev dev, const char *cd, const char *name);
extern OMstatus OMputSymbolN(OMdev dev, const char *cd, int clen, const char *name, int nlen);
extern OMstatus OMputApp(OMdev dev);
extern OMstatus OMputEndApp(OMdev dev);
extern OMstatus OMputAttr(OMdev dev);
extern OMstatus OMputEndAttr(OMdev dev);
extern OMstatus OMputAtp(OMdev dev);
extern OMstatus OMputEndAtp(OMdev dev);
extern OMstatus OMputBind(OMdev dev);
extern OMstatus OMputEndBind(OMdev dev);
extern OMstatus OMputBVar(OMdev dev);
extern OMstatus OMputEndBVar(OMdev dev);
extern OMstatus OMputObject(OMdev dev);
extern OMstatus OMputEndObject(OMdev dev);
extern OMstatus OMputError(OMdev dev);
extern OMstatus OMputEndError(OMdev dev);
extern OMstatus OMputComment(OMdev dev, const char *comment);
extern OMstatus OMputCommentN(OMdev dev, const char *comment, int len);

/* OMgetType
 * Get the type of the current token on device <dev>/
 * dev: device to look at.
 * type: where to store returned type.
 * return: 0 or some error code
 */
extern OMstatus OMgetType(OMdev dev, OMtokenType * type);

/* OMgetLength
 * Get the current token length.
 * dev: device to read from
 * len: where to put the token length
 * the last \\
 * (rem: for WCString it is the number of bytes not the number of
 * wide chars)
 * return: 0 or some error code
 */
extern OMstatus OMgetLength(OMdev dev, int *len);

/* OMgetSymbolLength
 * Get the current token (wich is assumed to be a symbol) lengths.
 * dev: device to read from
 * clen: where to put the cd length (not counting the last \\
 * nlen: where to put the name length (not counting the last \\
 * return: 0 or some error code
 */
extern OMstatus OMgetSymbolLength(OMdev dev, int *clen, int *nlen);

/* OMGet* functions.
 * They all take a device <dev> to get token from.
Some of them need more parameters to fill with the token content. They are thoroughly documented in OpenMath Specification shipped with the library.

* return: a status that reflect the operation success.

*/
extern OMstatus OMgetInt32(OMdev dev, int *i);
extern OMstatus OMgetFloat64(OMdev dev, double *d);
extern OMstatus OMgetBigInt(OMdev dev, char **data, int *len, int *sign, OMbigIntType * format);
extern OMstatus OMgetBigIntN(OMdev dev, char *data, int len, int *sign, OMbigIntType * format);
extern OMstatus OMgetByteArray(OMdev dev, char **data, int *len);
extern OMstatus OMgetByteArrayN(OMdev dev, char *data, int len);

/* OMgetString*
Beware! You are not supposed to use these functions unless you know for sure you are reading plain 8bits strings. Thus it is here only for speed/space consideration in very specific applications.
* If input is a 16 bit char string and you read it with these functions you will lose the 8 most significant bits of each char.
* You should rather refer to OMgetWCString* functions.
*/
extern OMstatus OMgetString(OMdev dev, char **str);
extern OMstatus OMgetStringN(OMdev dev, char *str, int len);

/* OMgetWCString*
These functions return 16 bits wide strings. (regardless input was done in 8 or 16 bits mode).
* Thus, most if not all applications should use these functions preferably to OMgetString*.
*/
extern OMstatus OMgetWCString(OMdev dev, OMUCS2 ** wcstr);
extern OMstatus OMgetWCStringN(OMdev dev, OMUCS2 * wcstr, int len);
extern OMstatus OMgetVar(OMdev dev, char **var);
extern OMstatus OMgetVarN(OMdev dev, char *var, int len);
extern OMstatus OMgetSymbol(OMdev dev, char **cd, char **name);
extern OMstatus OMgetSymbolN(OMdev dev, char *cd, int clen, char *name, int nlen);
extern OMstatus OMgetApp(OMdev dev);
extern OMstatus OMgetEndApp(OMdev dev);
extern OMstatus OMgetAttribute(OMdev dev);
extern OMstatus OMgetEndAttr(OMdev dev);
extern OMstatus OMgetAtp(OMdev dev);
extern OMstatus OMgetEndAtp(OMdev dev);
extern OMstatus OMgetBind(OMdev dev);
extern OMstatus OMgetEndBind(OMdev dev);
extern OMstatus OMgetBVar(OMdev dev);
extern OMstatus OMgetEndBVar(OMdev dev);
extern OMstatus OMgetObject(OMdev dev);
extern OMstatus OMgetEndObject(OMdev dev);
extern OMstatus OMgetError(OMdev dev);
extern OMstatus OMgetEndError(OMdev dev);
extern OMstatus OMgetComment(OMdev dev, char **comment);
extern OMstatus OMgetCommentN(OMdev dev, char *comment, int len);
/* OMbeginObject */
* Must be called before every new OpenMath object put.
* (Not before every token!)
* dev: device where new object is to be put.
* return: status describing operation success
*/
extern OMstatus OMbeginObject(OMdev dev);
/* OMendObject */
* Must be called after every OpenMath object put.
* (Not after every token!)
* dev: device where object has been put.
* return: status describing operation success
*/
extern OMstatus OMendObject(OMdev dev);
/* OMignoreComment */
* Set behavior of a device concerning comments.
* (Comments on an input device may safely be ignored.)
* dev: device to modify
* set: If set == OMtrue then device will ignore incoming comments
* If set == OMfalse then device will process incoming comments
* like other tokens.
* By default comments are ignored.
* Whatever is <set> value, output of comments is always done.
* return: previous value
*/
extern OMbool OMignoreComment(OMdev dev, OMbool set);
/* OMtokenCount */
* Reports the numbe rof tokens that have been in/output on a device
* dev: device to examine
* inTokenNb: where to store number of input tokens (if not NULL)
* outTokenNb: where to store number of output tokens (if not NULL)
*/
extern void OMtokenCount(OMdev dev, int *inTokenNb, int *outTokenNb);
/* OMgetDeviceEncoding */
* Get the currnet encoding used by a device
* dev: device to examine
* return: current encoding
*/
extern OMencodingType OMgetDeviceEncoding(OMdev dev);
/* OMsetDeviceEncoding */
* Set the encoding that will be used on a device
* BEWARE: changing encoding on a device that has already been used
* for IO is unsafe.
* but seting encoding on a new device is safe.
(in some occasions, it is not easy to know which encoding to use at device creation)
* dev: device to modify
* encoding: encoding to use
*
extern void OMsetDeviceEncoding(OMdev dev, OMencodingType encoding);
/* OMmakeDevice
* Create a device from a low level IO
* Warning: "IO" should be a "generated" (new) structure as it contains some
* state that is private to the device. It is very dangerous for two devices
* to share the same "IO" structure.
* encoding: encoding scheme used by device
* IO: low level I/O suport for device
* return: a newly allocated device
*/
extern OMdev OMmakeDevice(OMencodingType encoding, OMIO IO);
/* OMcloseDevice
* Close a device previously created with OMmakeDevice
* (embedded IO is closed too)
* dev: device to close
*/
extern void OMcloseDevice(OMdev dev);
/* OMmakeIOFd
* Create a low level IO object from a file descriptor.
* (May be used on socket for instance.)
* fd: file descriptor to wrap into the OpenMath IO object.
* return: a newly allocated IO object.
*/
extern OMIO OMmakeIOFd(int fd);
/* OMmakeIODevice
* Create a low level IO object from a FILE*.
* (May be used on stdin for instance.)
* fd: FILE* to wrap into the OpenMath IO object.
* return: a newly allocated IO object.
*/
extern OMIO OMmakeIODevice(FILE * f);
/* OMmakeIOString
* Create a low level IO object from a string (NUL terminator is not needed).
* (May be used for copy/paste for instance.)
* s: pointer to string to use into the OpenMath IO object.
* - In case of input device the string must be NUL terminated.
* - In case of output device string may be reallocated
* to fit size of outcoming objects.
* return: a newly allocated IO object.
*/
extern OMIO OMmakeIOString(char **s);
/* OMstatusToString
* Convert a status to a human readable string that explain its meaning
* status: status to explain
* return: corresponding string
extern char *OMstatusToString(OMstatus status);
/* OMtokenTypeToString
   Convert a tokenType to a human readable string
   * ttype: type to convert
   * return: corresponding string
*/
extern char *OMtokenTypeToString(OMtokenType ttype);
/* OMsetVerbosityLevel
   When using API some infos may be loged.
   * This set the required verbosity level.
   * level: level of verbosity.
   * 0 means nothing is neither printed
   * 1 everything is printed (default)
   * 2,... less verbose
   * return: last verbosity level
*/
extern int OMsetVerbosityLevel(int level);
/* OMsetVerbosityOutput
   When using API some infos may be loged.
   * This set the destination for logs.
   * logFile: where to output logs (default is stderr)
   * return: last output
*/
extern FILE *OMsetVerbosityOutput(FILE * logFile);
/* OMlibDynamicInfo
   Gather some informations about lib that can't be statically determined.
   * Complete them with some relevant static information too.
   * return: a newly allocated string
*/
extern char *OMlibDynamicInfo(void);
/** End Prototypes */
/* end of automaticaly updated part */

#ifdef WIN32
#include "windows.h"
/* OMmakeIOHandle
   Create a low level IO object from a widows handle.
   * handle: windows handle to wrap into the OpenMath IO object.
   * return: a newly allocated IO object.
*/
extern OMIO OMmakeIOHandle(HANDLE handle);
extern void OMfreeIOHandle(OMIO io);
#endif

#ifndef OM_DEV
/* The prototypes above are in fact collected from all these .h files */
#include "OMBase.h"
#include "OMdev.h"
#endif
39.15 Axiom OpenMath stub functions

These stub functions will eventually be expanded to handle OpenMath. See the OpenMath-Device domain in Volume 10.3. Note that the argument list for the Spad functions does not always match the argument list specified in the OpenMath specification.

There are 4 known OpenMath encodings which are set up in the OpenMathEncoding domain in Volume 10.3.

- Unknown
- Binary
- XML
- SGML

Axiom specific functions

This is used in OpenMathPackage in Volume 10.4.

(read OMdev) -> LispObject
(listCDs) -> List(String)
(listSymbols) -> List(String)
(supportsCD cd) -> Boolean
(supportsSymbol cd name) -> Boolean

defun om-Read

Read an OpenMath object from dev.

— defun om-Read —
(defun om-Read (dev)
  (declare (ignore dev)))

defun om-listCDs
Lists all of the CDs supported by Axiom.

  — defun om-listCDs —

(defun om-listCDs ()

  — defun om-listSymbols —

(defun om-listSymbols ()

  — defun om-supportsCD —

(defun om-supportsCD (cd)
  (declare (ignore cd)))

  — defun om-supportsSymbol —

(defun om-supportsSymbol (cd name)
  (declare (ignore cd name)))
Lisp conversion functions

The Lisp conversion functions are:

- `(toDev LispObject) -> OMdev`
- `(fromDev OMdev) -> LispObject`
- `(toStatus LispObject) -> LispObject`
- `(fromStatus OMstatus) -> LispObject`
- `(toEncodingType LispObject) -> OMencodingType`
- `(fromEncodingType OMencodingType) -> LispObject`
- `(toBigNumStr LispObject) -> char *`
- `(fromBigNumStr char *, int, int, OMbigIntType) -> LispObject`
- `(toConn LispObject) -> OMconn`
- `(fromConn OMconn) -> LispObject`
- `(toCString LispObject) -> char **`
- `(fromCString char **) -> LispObject`
- `(lispStringFromCString LispObject) -> LispObject`
- `(cStringFromLispString LispObject) -> LispObject`

defun om-setDevEncoding

This sets the encoding used for reading or writeing OpenMath objects to or from dev to enc.

```
(defun om-setDevEncoding (dev enc)
  (declare (ignore dev enc)))
```

Device manipulation functions

- `(openFileDev LispObject, ints, ...) -> LispObject`
- `(openStrDev LispObject, LispObject, LispObject) -> LispObject`
- `(closeDev LispObject, LispObject) -> LispObject`

defun om-openFileDev

This opens file fname for reading or writing OpenMath objects. The mode can be “r” for read, “w” for write, or “a” for append.

```
(defun om-openFileDev (fname fmode enc)
  (declare (ignore fname fmode enc)))
```
defun om-openStringDev

This opens the string str for reading and writing OpenMath objects in encoding enc.

(defun om-openStringDev (str enc)
  (declare (ignore str enc)))

defun om-closeDev

This closes dev, flushing output if necessary.

(defun om-closeDev (dev)
  (declare (ignore dev)))

Connection manipulation functions

These are covered in the OpenMathConnection domain in Volume 10.3.

(makeConn LispObject, LispObject) -> LispObject
(closeConn LispObject, LispObject) -> LispObject
(getConnInDev LispObject, LispObject) -> LispObject
(getConnOutDev LispObject, LispObject) -> LispObject

defun om-makeConn

(defun om-makeConn (conn)
  (declare (ignore conn)))

defun om-closeConn

(defun om-closeConn)
(defun om-closeConn (conn)
  (declare (ignore conn)))

---

defun om-getConnInDev

---

defun om-getConnInDev

---

defun om-getConnOutDev

---

Client/Server functions

These are covered in the OpenMathConnection domain in Volume 10.3. See OMconn.h

(bindTCP LispObject, LispObject, LispObject) -> LispObject
(connectTCP LispObject, int, ...) -> LispObject

defun om-bindTCP

---

defun om-bindTCP

defun om-connectTCP

— defun om-connectTCP —

(defun om-connectTCP (conn host port)
  (declare (ignore conn host port)))

Device input/output functions

Most of these functions are in the OpenMathDevice domain in Volume 10.3. The only exception seems to be the om-stringPtrToString and om-stringToStringPtr functions which are called in the domains that export primitives. Currently these are:

- Complex (10.3)
- DoubleFloat (10.3)
- Float (10.3)
- Fraction (10.3)
- Integer (10.3)
- List (10.3)
- SingleInteger (10.3)
- String (10.3)
- Symbol (10.3)
- ExpressionToOpenMath (10.4)
- OpenMathPackage (10.4)

Note that putSymbol2 is not implemented.

(getApp        LispObject, LispObject) -> LispObject
(getAtp        LispObject, LispObject) -> LispObject
(getAttr       LispObject, LispObject) -> LispObject
(getBind       LispObject, LispObject) -> LispObject
(getBVar       LispObject, LispObject) -> LispObject
(getByteArray  LispObject, LispObject) -> LispObject
(getEndApp     LispObject, LispObject) -> LispObject
(getEndAtp     LispObject, LispObject) -> LispObject
(getEndAttr    LispObject, LispObject) -> LispObject
### Axiom OpenMath Stub Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>(getEndBind)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(getEndBVar)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(getError)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(getError)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(getFloat)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(getInt)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(getObject)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(getString)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(getSymbol)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(getType)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(getVar)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(putApp)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(putAtp)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(putAttr)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(putBind)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(putBVar)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(putByteArray)</td>
<td>LispObject, LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(putEndApp)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(putEndAtp)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(putEndAttr)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(putEndBind)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(putEndBVar)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(putEndError)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(putEndObject)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(putError)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(putFloat)</td>
<td>LispObject, LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(putInt)</td>
<td>LispObject, LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(putObject)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(putString)</td>
<td>LispObject, LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(putSymbol2)</td>
<td>LispObject, LispObject, int nargs, ...</td>
<td>LispObject</td>
</tr>
<tr>
<td>(putVar)</td>
<td>LispObject, LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(stringPtrToString)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
<tr>
<td>(stringToStringPtr)</td>
<td>LispObject, LispObject</td>
<td>LispObject</td>
</tr>
</tbody>
</table>

---

#### defun om-getApp

Reads a begin application token from dev.

---

(defun om-getApp (dev)
  (declare (ignore dev)))
defun om-getAtp
Reads a begin attribute pair token from dev.
— defun om-getAtp —

(defun om-getAtp (dev)
  (declare (ignore dev)))

defun om-getAttr
Reads a begin attribute token from dev
— defun om-getAttr —

(defun om-getAttr (dev)
  (declare (ignore dev)))

defun om-getBind
Reads a begin binder token from dev.
— defun om-getBind —

(defun om-getBind (dev)
  (declare (ignore dev)))

defun om-getBVar
Reads a begin bound variable list token from dev.
— defun om-getBVar —

(defun om-getBVar (dev)
  (declare (ignore dev)))
defun om-getByteArray
Reads a byte array from dev.

(defun om-getByteArray (dev))

defun om-getEndApp
Reads an end application token from dev.

(defun om-getEndApp (dev)
  (declare (ignore dev)))

defun om-getEndAtp
Reads an end attribute pair token from dev.

(defun om-getEndAtp (dev)
  (declare (ignore dev)))

defun om-getEndAttr
Reads an end attribute token from dev.

(defun om-getEndAttr (dev)
  (declare (ignore dev)))
 CHAPTER 39. OPENMATH

defun om-getEndBind
Reads an end binder token from dev.

(defun om-getEndBind (dev)
  (declare (ignore dev)))


defun om-getEndBVar
Reads an end bound variable list token from dev.

(defun om-getEndBVar (dev)
  (declare (ignore dev)))


defun om-getEndError
Reads an end error token from dev.

(defun om-getEndError (dev)
  (declare (ignore dev)))


defun om-getEndObject
Reads an end object token from dev.

(defun om-getEndObject (dev)
  (declare (ignore dev)))
defun om-getError
Reads a begin error token from dev.
— defun om-getError —
(defun om-getError (dev)
  (declare (ignore dev)))

defun om-getFloat
Reads a float from dev.
— defun om-getFloat —
(defun om-getFloat (dev)
  (declare (ignore dev)))

defun om-getInt
Reads an integer from dev.
— defun om-getInt —
(defun om-getInt (dev)
  (declare (ignore dev)))

defun om-getObject
Reads a begin object token from dev.
— defun om-getObject —
(defun om-getObject (dev)
  (declare (ignore dev)))
defun om(getString)
Reads a string from dev.
— defun om(getString) —

(defun om(getString) (dev)
 (declare (ignore dev)))

——

defun om(getSymbol)
Reads a symbol from dev.
— defun om(getSymbol) —

(defun om(getSymbol) (dev)
 (declare (ignore dev)))

——

defun om(getType)
Returns the type of the next object on dev.
— defun om(getType) —

(defun om(getType) (dev)
 (declare (ignore dev)))

——

defun om(getVar)
Reads a variable from dev.
— defun om(getVar) —

(defun om(getVar) (dev)
 (declare (ignore dev)))
defun om-putApp

Writes a begin application token to dev.

— defun om-putApp —

(defun om-putApp (dev)
  (declare (ignore dev)))

—

defun om-putAtp

This writes a begin application pair token to dev.

— defun om-putAtp —

(defun om-putAtp (dev)
  (declare (ignore dev)))

—

defun om-putAttr

This writes a begin attribute token to dev.

— defun om-putAttr —

(defun om-putAttr (dev)
  (declare (ignore dev)))

—

defun om-putBind

This writes a begin binder token to dev.

— defun om-putBind —

(defun om-putBind (dev)
  (declare (ignore dev)))
defun om-putBVar
This writes a begin bound variable list token to dev.
— defun om-putBVar —

(defun om-putBVar (dev)
 (declare (ignore dev)))

-----

defun om-putByteArray
This writes a byte array to dev.
— defun om-putByteArray —

(defun om-putByteArray (dev b)
 (declare (ignore dev b)))

-----

defun om-putEndApp
This writes an end application token to dev.
— defun om-putEndApp —

(defun om-putEndApp (dev)
 (declare (ignore dev)))

-----

defun om-putEndAtp
This writes an end attribute pair to dev.
— defun om-putEndAtp —

(defun om-putEndAtp (dev)
 (declare (ignore dev)))

-----
defun om-putEndAttr

This writes an end attribute token to dev.

(defun om-putEndAttr (dev)
  (declare (ignore dev)))

defun om-putEndBind

This writes an end binder token to dev.

(defun om-putEndBind (dev)
  (declare (ignore dev)))

defun om-putEndBVar

This writes an end bound variable list token to dev

(defun om-putEndBVar (dev)
  (declare (ignore dev)))

defun om-putEndError

This writes an end error token to dev

(defun om-putEndError (dev)
  (declare (ignore dev)))

— defun om-putEndAttr —

— defun om-putEndBind —

— defun om-putEndBVar —

— defun om-putEndError —
defun om-putEndObject
This writes an end object token to dev.
       — defun om-putEndObject —
(defun om-putEndObject (dev)
  (declare (ignore dev)))

defun om-putError
This writes a begin error token to dev.
       — defun om-putError —
(defun om-putError (dev)
  (declare (ignore dev)))

defun om-putFloat
This writes the float f to dev.
       — defun om-putFloat —
(defun om-putFloat (dev f)
  (declare (ignore dev f)))

defun om-putInt
This writes the integer i to dev
       — defun om-putInt —
(defun om-putInt (dev i)
  (declare (ignore dev i)))
defun om-putObject
This writes a begin object token to dev.

(defun om-putObject (dev)
  (declare (ignore dev)))

defun om-putString
This writes the string s to dev.

(defun om-putString (dev s)
  (declare (ignore dev s)))

defun om-putSymbol
This writes the symbol nm using semantics from cd to dev.

(defun om-putSymbol (dev cd nm)
  (declare (ignore dev cd nm)))

defun om-putVar
This writes the variable v to dev.

(defun om-putVar (dev v)
  (declare (ignore dev v)))
defun om-stringToStringPtr

This is used in the SingleInteger domain in Volume 10.3. This is supposed to return the
string from its address? It would appear to be a nop in lisp.

— defun om-stringToStringPtr —

(defun om-stringToStringPtr (str)
  (declare (ignore str)))

———

defun om-stringPtrToString

This is used in the SingleInteger domain in Volume 10.3. This is supposed to return the
string address from a string? It would appear to be a nop in lisp.

— defun om-stringPtrToString —

(defun om-stringPtrToString (str)
  (declare (ignore str)))

———
Chapter 40

NRLIB code.lisp support code

defun makeByteWordVec2

— defun makeByteWordVec2 0 —

(defun |makeByteWordVec2| (maxelement initialvalue)
  (let ((n (cond ((null initialvalue) 7) ('t maxelement))))
    (make-array (length initialvalue)
      :element-type (list 'mod (1+ n))
      :initial-contents initialvalue)))

—

defmacro spadConstant

— defmacro spadConstant 0 —

(defmacro |spadConstant| (dollar n)
  '(spadcall (svref ,dollar (the fixnum ,n))))

—
### Chapter 41

**Monitoring execution**

**MONITOR**

This file contains a set of functions for monitoring the execution of the functions in a file. It constructs a hash table that contains the function name as the key and monitor-data structures as the value.

The technique is to use a :cond parameter on trace to call the monitor-incr function to incr the count every time a function is called.

*monitor-table* | HASH TABLE
---|---
  is the monitor table containing the hash entries

*monitor-nrlibs* | LIST of STRING
---|---
  list of nrlib filenames that are monitored

*monitor-domains* | LIST of STRING
---|---
  list of domains to monitor-report (default is all exposed domains)

monitor-data | STRUCTURE
---|---
  is the defstruct name of records in the table
  name is the first field and is the name of the monitored function
  count contains a count of times the function was called
  monitorp is a flag that skips counting if nil, counts otherwise
  sourcefile is the name of the file that contains the source code

***** SETUP, SHUTDOWN ****

monitor-inittable () | FUNCTION
---|---
  creates the hashtable and sets *monitor-table*
  note that it is called every time this file is loaded

monitor-end () | FUNCTION
---|---
  unhooks all of the trace hooks

***** TRACE, UNTRACE *****
monitor-add (name &optional sourcefile)  FUNCTION
  sets up the trace and adds the function to the table

monitor-delete (fn)  FUNCTION
  untraces a function and removes it from the table

monitor-enable (&optional fn)  FUNCTION
  starts tracing for all (or optionally one) functions that are in the table

monitor-disable (&optional fn)  FUNCTION
  stops tracing for all (or optionally one) functions that are in the table

***** COUNTING, RECORDING *****

monitor-reset (&optional fn)  FUNCTION
  reset the table count for the table (or optionally, for a function)

monitor-incr (fn)  FUNCTION
  increments the count information for a function it is called by trace to increment the count

monitor-decr (fn)  FUNCTION
  decrements the count information for a function

monitor-info (fn)  FUNCTION
  returns the monitor-data structure for a function

***** FILE IO *****

monitor-write (items file)  FUNCTION
  writes a list of symbols or structures to a file

monitor-file (file)  FUNCTION
  will read a file, scan for defuns, monitor each defun
  NOTE: monitor-file assumes that the file has been loaded

***** RESULTS *****

monitor-results ()  FUNCTION
  returns a list of the monitor-data structures

monitor-untested ()  FUNCTION
  returns a list of files that have zero counts

monitor-tested (&optional delete)  FUNCTION
  returns a list of files that have nonzero counts optionally calling monitor-delete on those functions

***** CHECKPOINT/RESTORE *****

monitor-checkpoint (file)  FUNCTION
  save the *monitor-table* in a loadable form

monitor-restore (file)  FUNCTION
  restore a checkpointed file so that everything is monitored

***** ALGEBRA *****

monitor-autoload ()  FUNCTION
  traces autoload of algebra to monitor corresponding source files
NOTE: this requires the /spad/int/algebra directory

monitor-dirname (args) FUNCTION
expects a list of 1 libstream (loadvol's arglist) and monitors the source
this is a function called by monitor-autoload

monitor-nrlib (nrlib) FUNCTION
takes an nrlib name as a string (eg POLY) and returns a list of
monitor-data structures from that source file

monitor-report () FUNCTION
generate a report of the monitored activity for domains in
*monitor-domains*

monitor-spadfile (name) FUNCTION
given a spad file, report all nrlibs it creates
this adds each nrlib name to *monitor-domains* but does not
trace the functions from those domains

monitor-percent () FUNCTION
ratio of (functions executed)/(functions traced)

monitor-apropos (str) FUNCTION
given a string, find all monitored symbols containing the string
the search is case-insensitive. returns a list of monitor-data items

for example:
suppose we have a file "/u/daly/testmon.lisp" that contains:
(defun foo1 () (print 'foo1))
(defun foo2 () (print 'foo2))
(defun foo3 () (foo1) (foo2) (print 'foo3))
(defun foo4 () (print 'foo4))

an example session is:

; FIRST WE LOAD THE FILE (WHICH INITs *monitor-table*)

>(load "/u/daly/monitor.lisp")
Loading /u/daly/monitor.lisp
Finished loading /u/daly/monitor.lisp
T

; SECOND WE LOAD THE TESTMON FILE
>(load "/u/daly/testmon.lisp")
T

; THIRD WE MONITOR THE FILE
>(monitor-file "/u/daly/testmon.lisp")
monitoring "/u/daly/testmon.lisp"
NIL

; FOURTH WE CALL A FUNCTION FROM THE FILE (BUMP ITS COUNT)
>(foo1)
FOO1
FOO1
; AND ANOTHER FUNCTION (BUMP ITS COUNT)
>(foo2)

FOO2
FOO2

; AND A THIRD FUNCTION THAT CALLS THE OTHER TWO (BUMP ALL THREE)
>(foo3)

FOO1
FOO2
FOO3
FOO3

; CHECK THAT THE RESULTS ARE CORRECT

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 2 MONITORP T SOURCEFILE 
"/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO2 COUNT 2 MONITORP T SOURCEFILE 
"/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO3 COUNT 1 MONITORP T SOURCEFILE 
"/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE 
"/u/daly/testmon.lisp")

; STOP COUNTING CALLS TO FOO2

>(monitor-disable 'foo2)
NIL

; INVOKE FOO2 THRU FOO3

>(foo3)

FOO1
FOO2
FOO3
FOO3

; NOTICE THAT FOO1 AND FOO3 WERE BUMPED BUT NOT FOO2

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 3 MONITORP T SOURCEFILE 
"/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO2 COUNT 2 MONITORP NIL SOURCEFILE 
"/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO3 COUNT 2 MONITORP T SOURCEFILE 
"/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE 
"/u/daly/testmon.lisp")

; NOTICE THAT FOO1 AND FOO3 WERE BUMPED BUT NOT FOO2

>(monitor-disable 'foo2)
NIL

; INVOKE FOO2 THRU FOO3

>(foo3)

FOO1
FOO2
FOO3
FOO3

; NOTICE THAT FOO1 AND FOO3 WERE BUMPED BUT NOT FOO2

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 3 MONITORP T SOURCEFILE 
"/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO2 COUNT 2 MONITORP NIL SOURCEFILE 
"/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO3 COUNT 2 MONITORP T SOURCEFILE 
"/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE
"/u/daly/testmon.lisp")

; TEMPORARILY STOP ALL MONITORING

>(monitor-disable)
NIL

; CHECK THAT NOTHING CHANGES

>(foo3)

FOO1
FOO2
FOO3
FOO3

; NO COUNT HAS CHANGED

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 3 MONITORP NIL SOURCEFILE
   "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO2 COUNT 2 MONITORP NIL SOURCEFILE
   "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO3 COUNT 2 MONITORP NIL SOURCEFILE
   "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE
   "/u/daly/testmon.lisp")

; MONITOR ONLY CALLS TO FOO1

>(monitor-enable 'foo1)
T

; FOO3 CALLS FOO1

>(foo3)

FOO1
FOO2
FOO3
FOO3

; FOO1 HAS CHANGED BUT NOT FOO2 OR FOO3

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 4 MONITORP T SOURCEFILE
   "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO2 COUNT 2 MONITORP NIL SOURCEFILE
   "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO3 COUNT 2 MONITORP NIL SOURCEFILE
   "/u/daly/testmon.lisp")
"/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE
 "/u/daly/testmon.lisp")

; MONITOR EVERYBODY

>(monitor-enable)
NIL

; CHECK THAT EVERYBODY CHANGES

>(foo3)
FOO1
FOO2
FOO3
FOO3

; EVERYBODY WAS BUMPED

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 5 MONITORP T SOURCEFILE
 "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO2 COUNT 3 MONITORP T SOURCEFILE
 "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO3 COUNT 3 MONITORP T SOURCEFILE
 "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE
 "/u/daly/testmon.lisp")

; WHAT FUNCTIONS WERE TESTED?

>(monitor-tested)
(FOO1 FOO2 FOO3)

; WHAT FUNCTIONS WERE NOT TESTED?

>(monitor-untested)
(FOO4)

; UNTRACE THE WHOLE WORLD, MONITORING CANNOT RESTART

>(monitor-end)
NIL

; CHECK THE RESULTS

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 5 MONITORP T SOURCEFILE
 "/u/daly/testmon.lisp")
; CHECK THAT THE FUNCTIONS STILL WORK

>(foo3)

FOO1
FOO2
FOO3
FOO3

; CHECK THAT MONITORING IS NOT OCCURRING

>(monitor-results)

(#S(MONITOR-DATA NAME FOO1 COUNT 5 MONITORP T SOURCEFILE 
  "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO2 COUNT 3 MONITORP T SOURCEFILE 
  "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO3 COUNT 3 MONITORP T SOURCEFILE 
  "/u/daly/testmon.lisp")
#S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE 
  "/u/daly/testmon.lisp")

---

defvar *monitor-domains*

— initvars —

(defvar *monitor-domains* nil "a list of domains to report")

---

defvar *monitor-nrlibs*

— initvars —

(defvar *monitor-nrlibs* nil "a list of nrlibs that have been traced")

---
defvar *monitor-table*

— initvars —

(defvar *monitor-table* nil "a table of all of the monitored data")

— postvars —

(eval-when (eval load)
  (unless *monitor-table* (monitor-inittable)))

defstruct $monitor-data

— initvars —

(defstruct monitor-data name count monitorp sourcefile)

defstruct $libstream

— initvars —

(defstruct libstream mode dirname (indextable nil) (indexstream nil))

defun Initialize the monitor statistics hashtable

[*monitor-table* p1314]

— defun monitor-inittable 0 —
(defun monitor-initable ()
  "initialize the monitor statistics hashtable"
  (declare (special *monitor-table*))
  (setq *monitor-table* (make-hash-table)))

---

defun End the monitoring process, we cannot restart
[*monitor-table* p1314]

— defun monitor-end 0 —

(defun monitor-end ()
  "End the monitoring process. we cannot restart"
  (declare (special *monitor-table*))
  (maphash
   #'(lambda (key value)
       (declare (ignore value))
       (eval `(untrace ,key)))
   *monitor-table*)

---

defun Return a list of the monitor-data structures
[*monitor-table* p1314]

— defun monitor-results 0 —

(defun monitor-results ()
  "return a list of the monitor-data structures"
  (let (result)
    (declare (special *monitor-table*))
    (maphash
     #'(lambda (key value)
         (declare (ignore key))
         (push value result))
     *monitor-table*)
    (mapcar #'(lambda (x) (pprint x))
            (sort result #'string-lessp :key #'monitor-data-name)))))
defun Add a function to be monitored

(defun monitor-add (name &optional sourcefile)
  "add a function to be monitored"
  (declare (special *monitor-table*))
  (unless (fboundp name) (load sourcefile))
  (when (gethash name *monitor-table*)
    (monitor-delete name))
  (eval `(trace (,name :cond (progn (monitor-incr ',name) nil))))
  (setf (gethash name *monitor-table*)
    (make-monitor-data
      :name name :count 0 :monitorp t :sourcefile sourcefile)))

defun Remove a function being monitored

(defun monitor-delete (fn)
  "Remove a function being monitored"
  (declare (special *monitor-table*))
  (eval `(untrace ,fn))
  (remhash fn *monitor-table*))

defun Enable all (or optionally one) function for monitoring

(defun monitor-enable (&optional fn)
  "enable all (or optionally one) function for monitoring"
  (declare (special *monitor-table*))
  (if fn

### defun Disable all (optionally one) function for monitoring

[*monitor-table* p1314]

```lisp
(defun monitor-disable &optional fn)
  "disable all (optionally one) function for monitoring"
  (declare (special *monitor-table*))
  (if fn
    (progn
      (eval `(untrace ,fn))
      (maphash
        #'(lambda (key value)
           (declare (ignore value))
           (eval `(untrace ,key))
           (setf (monitor-data-monitorp (gethash key *monitor-table*)) nil))
        *monitor-table*))
  )
```

### defun Reset the table count for the table (or a function)

[*monitor-table* p1314]

```lisp
(defun monitor-reset &optional fn)
  "reset the table count for the table (or a function)"
  (declare (special *monitor-table*))
  (if fn
    (setf (monitor-data-count (gethash fn *monitor-table*)) 0)
  )
```
(maphash
  #'(lambda (key value)
      (declare (ignore value))
      (setf (monitor-data-count (gethash key *monitor-table*)) 0))
  *monitor-table*))

---

defun Incr the count of fn by 1

["monitor-table" p1314]

— defun monitor-incr 0 —

(defun monitor-incr (fn)
  "incr the count of fn by 1"
  (let (data)
    (declare (special *monitor-table*))
    (setq data (gethash fn *monitor-table*))
    (if data
      (incf (monitor-data-count data)) ;; change table entry by side-effect
      (warn "\"s is monitored but not in table..do (untrace \"s\" fn fn)))

---

defun Decr the count of fn by 1

["monitor-table" p1314]

— defun monitor-decr 0 —

(defun monitor-decr (fn)
  "decr the count of fn by 1"
  (let (data)
    (declare (special *monitor-table*))
    (setq data (gethash fn *monitor-table*))
    (if data
      (decf (monitor-data-count data)) ;; change table entry by side-effect
      (warn "\"s is monitored but not in table..do (untrace \"s\" fn fn)))

---
defun Return the monitor information for a function

[*monitor-table* p1314]

— defun monitor-info 0 —

(defun monitor-info (fn)
  "return the monitor information for a function"
  (declare (special *monitor-table*))
  (gethash fn *monitor-table*))

———

defun Hang a monitor call on all of the defuns in a file

[done p??]
[done p??]
[monitor-add p1316]

— defun monitor-file 0 —

(defun monitor-file (file)
  "hang a monitor call on all of the defuns in a file"
  (let (expr (package "BOOT"))
    (format t "monitoring ~s\%" file)
    (with-open-file (in file)
      (catch 'done
        (loop
          (setq expr (read in nil 'done))
          (when (eq expr 'done) (throw 'done nil))
          (if (and (consp expr) (eq (car expr) 'in-package))
            (if (and (consp (second expr)) (eq (first (second expr)) 'quote))
              (setq package (string (second (second expr)))))
            (setq package (second expr)))
          (when (and (consp expr) (eq (car expr) 'defun))
            (monitor-add (intern (string (second expr)) package) file))))))

———

defun Return a list of the functions with zero count fields

[*monitor-table* p1314]

— defun monitor-untested 0 —
(defun monitor-untested ()
"return a list of the functions with zero count fields"
(let (result)
  (declare (special *monitor-table*))
  (maphash
   #'(lambda (key value)
       (if (and (monitor-data-monitorp value) (= (monitor-data-count value) 0))
         (push key result)))
   *monitor-table*)
  (sort result #'string-lessp))))

---

(defun Return a list of functions with non-zero counts

[monitor-delete p1316]
[*monitor-table* p1314]

— defun monitor-tested 0 —

(defun monitor-tested (&optional delete)
"return a list of functions with non-zero counts, optionally deleting them"
(let (result)
  (declare (special *monitor-table*))
  (maphash
   #'(lambda (key value)
       (when (and (monitor-data-monitorp value)
                  (> (monitor-data-count value) 0))
         (when delete (monitor-delete key))
         (push key result)))
   *monitor-table*)
  (sort result #'string-lessp))))

---

(defun Write out a list of symbols or structures to a file

— defun monitor-write 0 —

(defun monitor-write (items file)
"write out a list of symbols or structures to a file"
(with-open-file (out file :direction :output)
  (dolist (item items)
    (if (symbolp item)
defun Save the *monitor-table* in loadable form
[*monitor-table* p1314]
[*print-package* p??]

— defun monitor-checkpoint 0 —

(defun monitor-checkpoint (file)
  "save the *monitor-table* in loadable form"
  (let ((*print-package* t))
    (declare (special *print-package* *monitor-table*))
    (with-open-file (out file :direction :output)
      (format out "(in-package "BOOT")~%"
              (format out "(monitor-inittable)~%"
                      (dolist (data (monitor-results))
                        (format out "(monitor-add '~s ~s)~%"
                                (monitor-data-name data)
                                (monitor-data-sourcefile data))
                        (format out "(setf (gethash '~s *monitor-table*)
                                          (make-monitor-data :name '~s :count ~s :monitorp ~s
                                                             :sourcefile ~s))~%"
                                (monitor-data-name data)
                                (monitor-data-name data)
                                (monitor-data-count data)
                                (monitor-data-monitorp data)
                                (monitor-data-sourcefile data))))))

— defun monitor-restore 0 —

(defun monitor-restore (file)
  "restore a checkpointed file"
  (load file))
defunPrintinghelpdocumentation

—defunmonitor-help0—

(defun monitor-help ()
  (format t "\%";
  ;; MONITOR
  ;;
  ;; This file contains a set of function for monitoring the execution
  ;; of the functions in a file. It constructs a hash table that contains
  ;; the function name as the key and monitor-data structures as the value
  ;;
  ;; The technique is to use a :cond parameter on trace to call the
  ;; monitor-incr function to incr the count every time a function is called
  ;;
  ;; *monitor-table* HASH TABLE
  ;; is the monitor table containing the hash entries
  ;; *monitor-nrlibs* LIST of STRING
  ;; list of nrlib filenames that are monitored
  ;; *monitor-domains* LIST of STRING
  ;; list of domains to monitor-report (default is all exposed domains)
  ;; monitor-data STRUCTURE
  ;; is the defstruct name of records in the table
  ;; name is the first field and is the name of the monitored function
  ;; count contains a count of times the function was called
  ;; monitorp is a flag that skips counting if nil, counts otherwise
  ;; sourcefile is the name of the file that contains the source code
  ;;
  ;; ***** SETUP, SHUTDOWN *****
  ;;
  ;; monitor-inittable () FUNCTION
  ;; creates the hashtable and sets *monitor-table*
  ;; note that it is called every time this file is loaded
  ;; monitor-end () FUNCTION
  ;; unhooks all of the trace hooks
  ;;
  ;; ***** TRACE, UNTRACE *****
  ;;
  ;; monitor-add (name &optional sourcefile) FUNCTION
  ;; sets up the trace and adds the function to the table
  ;; monitor-delete (fn) FUNCTION
  ;; untraces a function and removes it from the table
  ;; monitor-enable (&optional fn) FUNCTION
  ;; starts tracing for all (or optionally one) functions that
  ;; are in the table
  ;; monitor-disable (&optional fn) FUNCTION


stops tracing for all (or optionally one) functions that
are in the table

***** COUNTING, RECORDING *****

monitor-reset (&optional fn) function
reset the table count for the table (or optionally, for a function)

monitor-incr (fn) function
increments the count information for a function
it is called by trace to increment the count

monitor-decr (fn) function
decrements the count information for a function

monitor-info (fn) function
returns the monitor-data structure for a function

***** FILE IO *****

monitor-write (items file) function
writes a list of symbols or structures to a file

monitor-file (file) function
will read a file, scan for defuns, monitor each defun
NOTE: monitor-file assumes that the file has been loaded

***** RESULTS *****

monitor-results () function
returns a list of the monitor-data structures

monitor-untested () function
returns a list of files that have zero counts

monitor-tested (&optional delete) function
returns a list of files that have nonzero counts
optionally calling monitor-delete on those functions

***** CHECKPOINT/RESTORE *****

monitor-checkpoint (file) function
save the *monitor-table* in a loadable form
monitor-restore (file) function
restore a checkpointed file so that everything is monitored

***** ALGEBRA *****

monitor-autoload () function
traces autoload of algebra to monitor corresponding source files
NOTE: this requires the /spad/int/algebra directory

monitor-dirname (args) function
expects a list of 1 libstream (loadvol's arglist) and monitors the source
this is a function called by monitor-autoload

monitor-nrlib (nrlib) function
takes an nrlib name as a string (eg POLY) and returns a list of
;;; monitor-data structures from that source file
;;; monitor-report () FUNCTION
;;; generate a report of the monitored activity for domains in
;;; *monitor-domains*
;;; monitor-spadfile (name) FUNCTION
;;; given a spad file, report all nrlibs it creates
;;; this adds each nrlib name to *monitor-domains* but does not
;;; trace the functions from those domains
;;; monitor-percent () FUNCTION
;;; ratio of (functions executed)/(functions traced)
;;; monitor-apropos (str) FUNCTION
;;; given a string, find all monitored symbols containing the string
;;; the search is case-insensitive. returns a list of monitor-data items
" ) nil)

Monitoring algebra files

defun Monitoring algebra code.lsp files

[*monitor-nrlibs* p1313]

— defun monitor-dirname 0 —

(defun monitor-dirname (args)
  "expects a list of 1 libstream (loadvol's arglist) and monitors the source"
  (let (name)
    (declare (special *monitor-nrlibs*))
    (setq name (libstream-dirname (car args)))
    (setq name (file-namestring name))
    (setq name (concatenate 'string " /spad/int/algebra/" name "/code.lsp"))
    (when (probe-file name)
      (push name *monitor-nrlibs*)
      (monitor-file name))))

— defun Monitor autoloaded files —

— defun monitor-autoload 0 —

(defun monitor-autoload ()


"traces autoload of algebra to monitor corresponding source files"
(trace (vmlisp::loadvol
 :entrycond nil
 :exitcond (progn (monitor-dirname system::arglist) nil))))

---

defun Monitor an nrlib

[*monitor-table* p1314]

---

defun monitor-nrlib 0 ---

(defun monitor-nrlib (nrlib)
 "takes an nrlib name as a string (eg POLY) and returns a list of
 monitor-data structures from that source file"
(let (result)
 (declare (special *monitor-table*))
 (maphash
   #'(lambda (k v)
       (declare (ignore k))
       (when (string= nrlib
         (pathname-name (car (last
           (pathname-directory (monitor-data-sourcefile v))))))
         (push v result)))
   *monitor-table*)
 result))

---

defun Given a monitor-data item, extract the nrlib name

---

defun monitor-libname 0 ---

(defun monitor-libname (item)
 "given a monitor-data item, extract the nrlib name"
 (pathname-name (car (last
   (pathname-directory (monitor-data-sourcefile item)))))

---
defun Is this an exposed algebra function?

— defun monitor-exposedp 0 —

(defun monitor-exposedp (fn)
  "exposed functions have more than 1 semicolon. given a symbol, count them"
  (> (count #\; (symbol-name fn)) 1))

——

defun Monitor exposed domains

TPDHERE: note that the file interp.exposed no longer exists. The exposure information is now in this book. This needs to work off the internal exposure list, not the file.

[done p??]
[done p??]
[*monitor-domains* p1313]

— defun monitor-readinterp 0 —

(defun monitor-readinterp ()
  "read interp.exposed to initialize *monitor-domains* to exposed domains. this is the default action. adding or deleting domains from the list will change the report results"
  (let (skip expr name)
    (declare (special *monitor-domains*))
    (setq *monitor-domains* nil)
    (with-open-file (in "/spad/src/algebra/interp.exposed")
      (read-line in)
      (read-line in)
      (read-line in)
      (read-line in)
      (catch 'done
        (loop
          (setq expr (read-line in nil "done"))
          (when (string= expr "done") (throw 'done nil))
          (cond
            ((string= expr "basic") (setq skip nil))
            ((string= expr "categories") (setq skip t))
            ((string= expr "hidden") (setq skip t))
            ((string= expr "defaults") (setq skip nil)))
          (when (and (not skip) (> (length expr) 58))
            (setq name (subseq expr 58 (length expr)))
            (setq name (string-right-trim #'(#\space) name)))
      ))
    ))
(when (> (length name) 0)
  (push name *monitor-domains*)))))

(defun Generate a report of the monitored domains

[monitor-readinterp p1326]
[*monitor-domains* p1313]

— defun monitor-report 0 —

(defun monitor-report ()
  "generate a report of the monitored activity for domains in *monitor-domains*
  (let (nrlibs nonzero total)
    (declare (special *monitor-domains*))
    (unless *monitor-domains* (monitor-readinterp))
    (setq nonzero 0)
    (setq total 0)
    (maphash #'(lambda (k v)
      (declare (ignore k))
      (let (nextlib point)
        (when (> (monitor-data-count v) 0) (incf nonzero))
        (incf total)
        (setq nextlib (monitor-libname v))
        (setq point (member nextlib nrlibs :test #'string= :key #'car))
        (if point
          (setf (cdr (first point)) (cons v (cdr (first point))))
          (push (cons nextlib (list v)) nrlibs))))
    *monitor-table*)
  (format t "~d of ~d (~d percent) tested~%" nonzero total
    (round (/ (* 100.0 nonzero) total)))
  (setq nrlibs (sort nrlibs #'string< :key #'car))
  (dolist (pair nrlibs)
    (let ((exposedcount 0) (testcount 0))
      (when (member (car pair) *monitor-domains* :test #'string=)
        (format t "for library ~s" (car pair))
        (dolist (item (sort (cdr pair) #'> :key #'monitor-data-count))
          (when (monitor-exposedp (monitor-data-name item))
            (incf exposedcount)
            (when (> (monitor-data-count item) 0) (incf testcount))
            (format t "~5d ~s" (monitor-data-count item)
              (monitor-data-name item))
            (if (= exposedcount testcount)
              (format t "a has all exposed functions tested" (car pair)))

```
defun Parse an )abbrev expression for the domain name

— defun monitor-parse 0 —

(defun monitor-parse (expr)
  (let (point1 point2)
    (setq point1 (position #\space expr :test #'char=))
    (setq point1 (position #\space expr :start point1 :test-not #'char=))
    (setq point1 (position #\space expr :start point1 :test #'char=))
    (setq point1 (position #\space expr :start point1 :test-not #'char=))
    (setq point2 (position #\space expr :start point1 :test #'char=))
    (subseq expr point1 point2)))

defun Given a spad file, report all nrlibs it creates

[done p??]
[done p??]
[monitor-parse p1328]
[*monitor-domains* p1313]

— defun monitor-spadfile 0 —

(defun monitor-spadfile (name)
  "given a spad file, report all nrlibs it creates"
  (let (expr)
    (declare (special *monitor-domains*))
    (with-open-file (in name)
      (catch 'done
        (loop
          (setq expr (read-line in nil 'done))
          (when (eq expr 'done) (throw 'done nil))
          (when (and (> (length expr) 4) (string= (subseq expr 0 4) ")abb")(setq *monitor-domains*
            (adjoin (monitor-parse expr) *monitor-domains* :test #'string=))))))))
defun Print percent of functions tested
[*monitor-table* p1314]

— defun monitor-percent 0 —

(defun monitor-percent ()
  "Print percent of functions tested"
  (let (nonzero total)
    (declare (special *monitor-table*))
    (setq nonzero 0)
    (setq total 0)
    (maphash #'(lambda (k v)
                  (declare (ignore k))
                  (when (> (monitor-data-count v) 0) (incf nonzero))
                  (incf total))
      *monitor-table*)
    (format t "~d of ~d (~d percent) tested~%" nonzero total
      (round (/ (* 100.0 nonzero) total))))

———

defun Find all monitored symbols containing the string
[*monitor-table* p1314]

— defun monitor-apropos 0 —

(defun monitor-apropos (str)
  "given a string, find all monitored symbols containing the string
  the search is case-insensitive. returns a list of monitor-data items"
  (let (result)
    (maphash #'(lambda (k v)
                  (when
                    (search (string-upcase str)
                      (string-upcase (symbol-name k))
                      :test #'string=)
                    (push v result)))
      *monitor-table*
    result))

———
Chapter 42

HyperDoc Basic Command support

42.1 Hyperdoc macro handling and util.ht

All of the macros used in hyperdoc are in this hash table. User-defined macros are read from the file \texttt{doc/util.ht}

\texttt{defvar $htMacroTable}

\begin{verbatim}
— initvars —

(defvar $htMacroTable (make-hash-table :test #'equal))

—

These are the primitive hyperdoc commands. They are directly implemented. The \texttt{build-HtMacroTable} function adds these to the \texttt{$htMacroTable} at startup.

\texttt{defvar $primitiveHtCommands}

\begin{verbatim}
— initvars —

(defvar $primitiveHtCommands
  '(((\ContinueButton" . 1)
     \("\andexample" . 1)
CHAPTER 42. HYPERDOC BASIC COMMAND SUPPORT

\begin{itemize}
\item \texttt{\textbackslash \texttt{\textbackslash autobutt}} . 0
\item \texttt{\textbackslash \texttt{\textbackslash autobuttons}} . 0
\item \texttt{\textbackslash \texttt{\textbackslash begin}} . 1
\item \texttt{\textbackslash \texttt{\textbackslash beginscroll}} . 0
\item \texttt{\textbackslash \texttt{\textbackslash bound}} . 1
\item \texttt{\textbackslash \texttt{\textbackslash hbox}} . 1
\item \texttt{\textbackslash \texttt{\textbackslash centerline}} . 1
\item \texttt{\textbackslash \texttt{\textbackslash downlink}} . 2
\item \texttt{\textbackslash \texttt{\textbackslash em}} . 0
\item \texttt{\textbackslash \texttt{\textbackslash end}} . 1
\item \texttt{\textbackslash \texttt{\textbackslash endscroll}} . 0
\item \texttt{\textbackslash \texttt{\textbackslash example}} . 1
\item \texttt{\textbackslash \texttt{\textbackslash free}} . 1
\item \texttt{\textbackslash \texttt{\textbackslash graphpaste}} . 1
\item \texttt{\textbackslash \texttt{\textbackslash helppage}} . 1
\item \texttt{\textbackslash \texttt{\textbackslash htbdmurl}} . 0
\item \texttt{\textbackslash \texttt{\textbackslash htmfile}} . 1
\item \texttt{\textbackslash \texttt{\textbackslash indent}} . 1
\item \texttt{\textbackslash \texttt{\textbackslash inputbitmap}} . 1
\item \texttt{\textbackslash \texttt{\textbackslash inputstring}} . 3
\item \texttt{\textbackslash \texttt{\textbackslash item}} . 0
\item \texttt{\textbackslash \texttt{\textbackslash keyword}} . 1
\item \texttt{\textbackslash \texttt{\textbackslash link}} . 2
\item \texttt{\textbackslash \texttt{\textbackslash lispdownlink}} . 2
\item \texttt{\textbackslash \texttt{\textbackslash lispmemolink}} . 2
\item \texttt{\textbackslash \texttt{\textbackslash lispwindowlink}} . 2
\item \texttt{\textbackslash \texttt{\textbackslash menudownlink}} . 2
\item \texttt{\textbackslash \texttt{\textbackslash menuitemstyle}} . 1
\item \texttt{\textbackslash \texttt{\textbackslash menulink}} . 2
\item \texttt{\textbackslash \texttt{\textbackslash menuulispdownlink}} . 2
\item \texttt{\textbackslash \texttt{\textbackslash menuulispmemolink}} . 2
\item \texttt{\textbackslash \texttt{\textbackslash menuulispwindowlink}} . 2
\item \texttt{\textbackslash \texttt{\textbackslash menumemolink}} . 2
\item \texttt{\textbackslash \texttt{\textbackslash menuwindowlink}} . 2
\item \texttt{\textbackslash \texttt{\textbackslash newline}} . 0
\item \texttt{\textbackslash \texttt{\textbackslash radioboxes}} . 3
\item \texttt{\textbackslash \texttt{\textbackslash space}} . 1
\item \texttt{\textbackslash \texttt{\textbackslash spadcommand}} . 1
\item \texttt{\textbackslash \texttt{\textbackslash stringvalue}} . 1
\item \texttt{\textbackslash \texttt{\textbackslash tab}} . 1
\item \texttt{\textbackslash \texttt{\textbackslash table}} . 1
\item \texttt{\textbackslash \texttt{\textbackslash vspace}} . 1
\item \texttt{\textbackslash \texttt{\textbackslash windowlink}} . 2
\end{itemize}
defun Build the table of hyperdoc macros

Hash user-defined macros from doc/util.ht into htMacroTable. Hash primitive hyperdoc macros into htMacroTable. [buildHtMacroTable util.ht (vol7.1)]

(defun buildHtMacroTable ()
  (let (fn)
    (declare (special $htMacroTable |$primitiveHtCommands|))
    (setq fn (concat (getenviron "AXIOM") "/doc/util.ht"))
    (cond
      ((probe-file fn)
       (with-open-file (instream fn)
         (loop
          for line = (read-line instream nil :eof)
          until (eq line :eof)
          do
            (when
              (multiple-value-bind (command numOfArgs) (getHtMacroItem line)
                (hput $htMacroTable command numOfArgs)))
          (dolist (pair |$primitiveHtCommands|)
            (hput $htMacroTable (car pair) (cdr pair)))
        (t (sayBrightly "Warning: macro table not found"))
        |$htMacroTable|))
    |

----

defun Get new command name and number of args

This processes newcommand lines read from doc/util.ht An example newcommand looks like

\newcommand{\menulink}[2] {\menuunlink{#1}{#2}}

This function returns a pair whose CAR is the new command name and whose CDR is the number of arguments. If there are zero arguments the brackets and number will not appear. However brackets can appear in the new command so we need to fix the original code to handle this new case. We set up a wall starting after the first closing brace. 
\newcommand{\beginmenu} \{\beginitems{\MenuDotBitmap}\}

[getHtMacroItem util.ht (vol7.1)]

getHtMacroItem : String → Values (String NonNegativeInteger)

\[\text{— defun getHtMacroItem —}\]

\[
\text{(defun getHtMacroItem (line)}
\text{\hspace{1em}(let (k command m i j wall digitString)}
\text{\hspace{1em}(when (stringPrefix? \"{\newcommand{\line} \}
\text{\hspace{1em}(setq wall (position \"{\line :start k)) ; wall off the body of command}
\text{\hspace{1em}(setq command (substring line 12 (- k 12)))}
\text{\hspace{1em}(setq m (length line))}
\text{\hspace{1em}(setq i (position \[ line :start k))}
\text{\hspace{1em}(if (and i (< i m) wall (< i wall))}
\text{\hspace{1em}(progn ; brackets. parse number of args}
\text{\hspace{1em}(setq j (position \[ line :start (+ i 1)))}
\text{\hspace{1em}(setq digitString (substring line (+ i 1) (- (- j i) 1)))}
\text{\hspace{1em}(when (every #'digitp digitString)}
\text{\hspace{1em}(values command (parse-integer digitString))))}
\text{\hspace{1em}(values command 0)))))) ; no brackets}
\]

We populate the htMacroTable at load time.

\[\text{— postvars —}\]

\[
\text{(eval-when (eval load)}
\text{\hspace{1em}(|buildHtMacroTable|))}
\]

defun Is the first string a prefix of the second?

\[\text{— defun stringPrefix? 0 —}\]

\[
\text{(defun stringPrefix? (pref str)}
\text{\hspace{1em}(let (lp)}
\text{\hspace{1em}(cond}
\text{\hspace{1em}((null (and (stringp pref) (stringp str))) nil)}
\text{\hspace{1em}((eq l (setq lp (length pref)) 0) t)}
\text{\hspace{1em}((> lp (length str)) nil)}
\text{\hspace{1em}(t (every #'char= pref str))))))}
\]
42.2 Functions creating pages

Most of the functions create a new page with a call to the function \( \texttt{htMakePage} \). This function takes an association list which has several possible keys.

- \texttt{domainConditions} with tests such as \((\text{isDomain} S \ (\text{String}))\) constraining the domains. The possible tests are
  - \texttt{isDomain}
- \texttt{text} which takes a string argument which may contain latex-like format strings.
  - a plain string
  - \texttt{beginmenu}
  - \texttt{blankline}
  - \texttt{centerline}
  - \texttt{em} with an argument to be emphasized
  - \texttt{indent} sets the column
  - \texttt{indentrel} does a relative indent by a positive or negative amount
- \texttt{inputStrings}
  - \texttt{item} occurs between a \texttt{beginmenu} and \texttt{endmenu} text
  - \texttt{lispdownlink} takes a string and a function to call
  - \texttt{lisplinks}
  - \texttt{menuitemstyle} takes a set of characters as an argument
  - \texttt{newline}
  - \texttt{space} with a numeric argument of the number of spaces
  - \texttt{tab} with a numeric argument indicating the tab column
  - \texttt{vspace} with the number of blank lines needed
- \texttt{bcStrings} which takes a list. The first element is the width of the input box, the second is the default contents, the third is the name of the variable to hold the contents, and the fourth is the domains allowed as input (see \texttt{domainConditions} above).
- \texttt{bcLinks} which takes a list containing strings and function calls. It will link to another page by calling the page generation function for that page.
- \texttt{doneButton} which takes 2 arguments, a label and a function to call.
- \texttt{radioButtons} takes a button name and set of lists, each one creating a new radio button
• inputStrings
• bcHt

The `htMakeDoneButton` will put a button on the page with the given title and a function to call when pressed.

```lisp
(defun Basic Command matrix entry
  ;; bcReadMatrix
  ;; htInitPage
  ;; htpSetProperty
  ;; htMakePage
  ;; htShowPage

  ;; This routine is called from several places to enter a matrix. The argument `bcReadMatrix` is
  ;; the name of a function to call when the matrix has been entered. This value is set as an
  ;; `exitFunction` in the page's association table.

  (defun bcMatrix () (bcReadMatrix nil))
)

(defun Read Matrix
  ;; bcReadMatrix
  ;; htInitPage
  ;; htpSetProperty
  ;; htMakePage
  ;; htShowPage

  ;; Enter the size of the matrix:
  ;; Number of rows: 2
  ;; Number of columns: 2

  ;; How would you like to enter the matrix?
  ;; ■ By entering individual entries
  ;; ■ By formula

  ;; This routine is called from several places to enter a matrix. The argument `bcReadMatrix` is
  ;; the name of a function to call when the matrix has been entered. This value is set as an
  ;; `exitFunction` in the page's association table.

  (defun bcReadMatrix (exitFunctionOrNil)
    (let (page)
      (setq page (htInitPage "Matrix Basic Command" nil))
      (htpSetProperty page 'exitFunction exitFunctionOrNil)
      (htMakePage
        '((domainConditions (isDomain PI (PositiveInteger)))))
  )
)
42.2. FUNCTIONS CREATING PAGES

---

**defun Input Matrix By Formula**

---

Pressing the Continue button will call the function \texttt{bcInputMatrixByFormulaGen} due to this line:
defun bcInputMatrixByFormula (htPage junk)

(declare (ignore junk))
(let (page nrows ncols)
 (declare (special $bcParseOnly$))
 (setq page (htInitPage "Basic Matrix Command" (htpPropertyList htPage)))
 (htShowPage))

'('((domainConditions |isDomain| S (Symbol))
 (isDomain| FE| (Expression (Integer))))
 (text . "Enter the \{row variable\}: ")
 (text . \\tab{36} (bcStrings (6 i |rowVar| S))
 (text . \blankline \newline )
 (text . "Enter the \{column variable\}: ")
 (text . \\tab{36} (bcStrings (6 j |colVar| S))
 (text . \blankline \newline )
 (text . \\menuitemstyle{}\tab{2})
 (text . "Enter the general \{formula\} for the entries:")
 (text . \\newline\tab{2} "(bcStrings (40 "1/(x - i - j - 1)" |formula| FE))))

(setq nrows
 (if (null $bcParseOnly$)
  (objValUnwrap (htpLabelSpadValue htPage 'rows)))
  (parse-integer (htpLabelInputString htPage 'rows))))

(setq ncols
 (if (null $bcParseOnly$)
  (objValUnwrap (htpLabelSpadValue htPage 'cols)))
  (parse-integer (htpLabelInputString htPage 'cols))))

(ht SetProperty page 'nrows nrows)
(ht SetProperty page 'ncols ncols)

(htShowPage)))

defun Basic Command Matrix by Formula generate

[htpProperty p1408]
[htpLabelInputString p1409]
[bcGen p1399]
[strconc p??]
defun bcInputMatrixByFormulaGen (htPage)
(let (fun formula rowVar colVar nrows ncols)
  (cond
   ((setq fun (htpProperty htPage '|exitFunction|))
    (funcall fun htPage))
   (t
    (setq formula (htpLabelInputString htPage '|formula|))
    (setq rowVar (htpLabelInputString htPage '|rowVar|))
    (setq colVar (htpLabelInputString htPage '|colVar|))
    (setq nrows (htpProperty htPage '|nrows|))
    (setq ncols (htpProperty htPage '|ncols|))
    (bcGen
     (strconc "matrix([[
        " formula
     " for " colVar
     " in 1.." (princ-to-string ncols)
     "] for " rowVar
     " in 1.." (princ-to-string nrows)
     "]")
    ))))

---

---

— defun bcInputMatrixByFormulaGen —

(defun |bcInputMatrixByFormulaGen| (htPage)
(let (fun formula rowVar colVar nrows ncols)
  (cond
   ((setq fun (htpProperty| htPage '|exitFunction|))
    (funcall fun htPage))
   (t
    (setq formula (htpLabelInputString| htPage '|formula|))
    (setq rowVar (htpLabelInputString| htPage '|rowVar|))
    (setq colVar (htpLabelInputString| htPage '|colVar|))
    (setq nrows (htpProperty| htPage '|nrows|))
    (setq ncols (htpProperty| htPage '|ncols|))
    (bcGen
     (strconc "matrix([[
        " formula
     " for " colVar
     " in 1.." (princ-to-string ncols)
     "] for " rowVar
     " in 1.." (princ-to-string nrows)
     "]")
    )))

---

---

defun Input Explicit Matrix

[objValUnwrap p148]
[htpLabelSpadValue p1410]
[htpLabelInputString p1409]
[parse-integer p??]
[length p??]
[nreverse0 p??]
[strconc p??]
[htInitPage p1415]
[htpPropertyList p1408]
[bcHt p1414]
[htMakePage p1416]
[htMakeDoneButton p1437]
[htpSetProperty p1408]
[htShowPage p1416]
[$EmptyMode p640]
[$bcParseOnly p1403]
CHAPTER 42. HYPERDOC BASIC COMMAND SUPPORT

Pressing the Continue button will call the function `bcGenExplicitMatrix` due to this line:

```lisp
(|htMakeDoneButton| "Continue" '|bcGenExplicitMatrix|)
```

---

```lisp
(defun |bcInputExplicitMatrix| (htPage junk)
  (declare (ignore junk))
  (let (nrows ncols cond wrows wcols rowpart colpart prefix k name
     labelList page t1 t2)
    (declare (special |$EmptyMode| |$bcParseOnly|))
    (setq nrows
      (if (null |$bcParseOnly|)
        (|objValUnwrap| (|htpLabelSpadValue| htPage '|rows|))
        (parse-integer (|htpLabelInputString| htPage '|rows|)))))
    (setq ncols
      (if (null |$bcParseOnly|)
        (|objValUnwrap| (|htpLabelSpadValue| htPage '|cols|))
        (parse-integer (|htpLabelInputString| htPage '|cols|)))))
    (setq k 0)
    (setq wrows (|#| (princ-to-string nrows)))
    (setq wcols (|#| (princ-to-string ncols)))
    (setq labelList
      (do ((i 1 (1+ i))) ((> i nrows) t1)
        (setq t1 nil)
        (setq t1
          (append t1
            (do ((j 1 (1+ j))) ((> j ncols) (nreverse0 t2))
              (setq t2 nil)
              (append t1
                (do ((j 1 (1+ j))) ((> j ncols) (nreverse0 t2))
                  (setq t2 nil)
                  (append t2
                    (progn
                      (setq rowpart (strconc "{\em Row" (|htStringPad| i wrows)))
                      (setq colpart (strconc ", Column" (|htStringPad| j wcols)
                          ":\space(2)"))
                      (setq prefix (strconc rowpart colpart))
                      (setq name (intern (princ-to-string (setq k (1+ k)))))
                      (list prefix "" 30 0 name 'P))
```
defun Basic Command generate explicit matrix

(defvar bcGenExplicitMatrix (lambda (htPage)  
  (let (fun)
    (htpSetProperty htPage 'matrix (htpInputAreaAlist htPage))
    (if (setq fun (htpProperty htPage 'exitFunction))
      (funcall fun htPage)
      (bcGen (bcMatrixGen htPage))))

---

defun Basic Command generate matrix

(defvar bcMatrixGen (lambda (htPage)  
  (let (fun)
    (assoc fun (htpProperty htPage 'exitFunction))
    (strconc (systemError fun) 
      (strconc (bcwords2liststring fun) 
        (htpProperty htPage 'matrix) 
        (strconc (htpInputAreaAlist htPage) 
          (strconc (bcGenExplicitMatrix fun) 
            (bcMatrixGen fun))))))

---
(defun |bcMatrixGen| (htPage)
  (let ((nrows ncols formula rowVar colVar mat k matform matstring)
    (setq nrows (htpProperty htPage '|nrows|))
    (setq ncols (htpProperty htPage '|ncols|))
    (setq mat (htpProperty htPage '|matrix|))
    (cond
      ((setq formula (lassoc '|formula| mat))
       (setq formula (elt formula 0))
       (setq rowVar (elt (lassoc '|rowVar| mat) 0))
       (setq colVar (elt (lassoc '|colVar| mat) 0))
       (strconc "matrix([" formula
         " for " colVar
         " in 1.." (princ-to-string ncols)
         "] for " rowVar
         " in 1.." (princ-to-string nrows)
         "]")
      )
      ((setq mat (htpProperty htPage '|matrix|))
       (setq mat (reverse mat))
       (setq k (- 1))
       (setq matform
         (loop for i from 0 to (1- nrows)
            collect (loop for j from 0 to (1- ncols)
                         collect (elt (elt mat (incf k)) 1))))
       (setq matstring
         (|bcwords2liststring|
          (loop for t1 in matform collect (|bcwords2liststring| t1))))
       (strconc "matrix(" matstring ")")
      (t (systemError nil))))

:-- Hyperdoc commands other than solve and matrix

;--- defun Basic Command iteration

[bcMatrixGen p1341]

--- defun bcDrawIt2 ---

(defun |bcDrawIt2| (ind a b)
  (strconc "\{\" ind "=\" a "\{\.. b "\{\")

:--
defun Indefinite Integration Basic Command

[htInitPage p1415]
[htMakePage p1416]
[htShowPage p1416]
[$EmptyMode p640]

Pressing the Continue button will call the function bcIndefiniteIntegrateGen due to this line:

```lisp
(defun bcIndefiniteIntegrate ()
  (declare (special $EmptyMode)))
```

---

(defun bcIndefiniteIntegrateGen ()
  (htInitPage)
  'Indefinite Integration Basic Command nil)
  (htMakePage)
  'domainConditions (isDomain EM $EmptyMode)
  (isDomain EM String)
  (isDomain SY Symbol))
  (text newline )
  (text \menuitemstyle{}\tab{2})
  (text Enter the {em function} you would like to integrate:)
  (text Enter the {em variable of integration}:)
  (bcStrings (45 "1/(x**2 + 6)" integrand EM))
  (text newline )
  (text \menuitemstyle{}\tab{2})
  (text Enter the {em variable of integration}:)
  (text \tab{37}) (bcStrings (10 x symbol SY))
  (doneButton "Continue" bcIndefiniteIntegrateGen)))
  (htShowPage)
defun bcIndefiniteIntegrateGen

(defun bcIndefiniteIntegrateGen (htPage)
  (let (integrand var)
    (setq integrand (htpLabelInputString htPage '|integrand|))
    (setq var (htpLabelInputString htPage '|symbol|))
    (bcGen (strconc "integrate(" integrand "," var ")"))))

defun Definite Integration Basic Command

Pressing the Continue button will call the function bcDefiniteIntegrateGen due to this line:

(defun bcDefiniteIntegrateGen (htPage)
  (let (integrand var)
    (setq integrand (htpLabelInputString htPage '|integrand|))
    (setq var (htpLabelInputString htPage '|symbol|))
    (bcGen (strconc "integrate(" integrand "," var ")"))))
42.2. FUNCTIONS CREATING PAGES

— defun bcDefiniteIntegrate —

(defun |bcDefiniteIntegrate| ()
 (declare (special |$EmptyMode|))
 (|htInitPage| '|Definite Integration Basic Command| NIL)
 (|htMakePage|
 '(((|domainConditions| (|isDomain| EM |$EmptyMode|)
     (|isDomain| S (|String|)) (|isDomain| SY (|Symbol|)))
   (|text| . "\newline ")
   (|text| . "\menuitemstyle{}\tab{2}"
   (|text| . "Enter the {\em function} you would like to integrate:"
   (|text| . "\newline\tab{2} ")
   (|bcStrings| (45 "1/(x**2 + 6)" |integrand| EM))
   (|text| . "%newline")
   (|text| . "\menuitemstyle{}}}\tab{2}"
   (|text| . "Enter the {\em variable of integration}:")
   (|text| . "\tab{37}"
   (|bcStrings| (10 |x| |symbol| SY))
   (|text| . "%newline")
   (|text| . "\menuitemstyle{}]\tab{2}"
   (|text| . "%newline Enter {\em lower limit}:")
   (|radioButtons| |fromButton|
   ("" "Minus infinity" |minusInfinity|
   (""
     (((|text| . "A finite point:\tab{15}"
     (|bcStrings| (10 0 |from| EM . |bcOptional|)))
   |fromPoint|))
   (|text| . "%newline")
   (|text| . "\menuitemstyle{}}}\tab{2}"
   (|text| . "\indent{2}\newline Enter {\em upper limit}:")
   (|radioButtons| |toButton|
   ("" "Plus infinity" |plusInfinity|
   (""
     (((|text| "A finite point:\tab{15}"
     (|bcStrings| (10 |y| |to| EM . |bcOptional|)))
   |toPoint|))
   (|doneButton| "Continue" |bcDefiniteIntegrateGen|)))
 (|htShowPage|))

defun bcDefiniteIntegrateGen

[|htpLabelInputString| p1409]
[|htpButtonValue| p1406]
[strconc p??]
[|bcGen| p1399]
CHAPTER 42. HYPERDOC BASIC COMMAND SUPPORT

— defun bcDefiniteIntegrateGen —

(defun |bcDefiniteIntegrateGen| (htPage)
  (let (integrand var lowerLimit upperLimit varpart)
    (setq integrand (|htpLabelInputString| htPage '|integrand|))
    (setq var (|htpLabelInputString| htPage '|symbol|))
    (setq lowerLimit
      (if (eq (|htpButtonValue| htPage '|fromButton|) '|fromPoint|)
          (|htpLabelInputString| htPage '|from|)
        "%minusInfinity"))
    (setq upperLimit
      (if (eq (|htpButtonValue| htPage '|toButton|) '|toPoint|)
          (|htpLabelInputString| htPage '|to|)
        "%plusInfinity"))
    (setq varpart (strconc var " = " lowerLimit ".." upperLimit))
    (|bcGen| (strconc "integrate(" integrand "," varpart ")"))))

— defun Sum Basic Command —

(defun |bcSum|
  (|htInitPage| p1415)
  (|htMakePage| p1416)
  (|htShowPage| p1416)
  (|$EmptyMode| p640)

  Enter the function you would like to sum:
  \[ i^{\#3} \]

  Enter the summation index:
  \[ i \]

  Enter the limits of the sum:
  From: \[ 1 \] To: \[ n \]

  Pressing the Continue button will call the function |bcSumGen| due to this line:

  (|doneButton| "Continue" |bcSumGen|)

— defun bcSum —
42.2. FUNCTIONS CREATING PAGES

(defun bcSum ()
  (declare (special $EmptyMode))
  (htInitPage 'Sum Basic Command NIL)
  (htMakePage
    '((domainConditions
        (isDomain EM $EmptyMode)
        (isDomain S (String))
        (isDomain SY (Symbol)))
      (text . "newline ")
      (text . "menuitemstyle{}\tab{2}"
        Enter the {em function} you would like to sum:")
      (text . "newline\tab{2}"
        (bcStrings (44 "i**3" summand EM))
        (text . "\blankline ") (text . "newline "
        (text . "menuitemstyle{}\tab{2}"
          Enter the {em summation index}:")
      (text . "\tab{36}"
        (bcStrings (10 i index SY))
      (text . "\blankline " )
      (text . "\newline"
        (text . "menuitemstyle{}\tab{2}"
          Enter the limits of the sum:")
      (text . "newline\tab{2}\em From:")
      (bcStrings (10 1 first S))
      (text . "\tab{36}\em To:" )
      (text . "\tab{36}"
        (bcStrings (10 n last S))
      (doneButton "Continue" bcSumGen)))
  (htShowPage)))

defun bcSumGen

[htpLabelInputString p1409]
[strconc p??]
[bcGen p1399]

 ----- defun bcSumGen -----

(defun bcSumGen (htPage)
  (let (mand index car last)
    (setq mand (htpLabelInputString htPage 'summand))
    (setq index (htpLabelInputString htPage 'index))
    (setq car (htpLabelInputString htPage 'first))
    (setq last (htpLabelInputString htPage 'last))
    (bcGen (strconc "sum(" mand "," index " = " car "." last ")"))))

-----
defun Sum Basic Command

(defun bcProduct ()
  (declare (special $EmptyMode))
  (htInitPage 'Product Basic Command NIL)
  (htMakePage
   '(((domainConditions (isDomain EM $EmptyMode)
   (isDomain S (String)) (isDomain SY (Symbol)))
   (text . "Enter the {\em function} you would like to compute the product of:"))
   (inputStrings ("" "5" "i**2" |mand| EM))
   (text . "\vspace{1}\newline Enter the {\em index of the product}:")
   (inputStrings ("" "" 10 \newline Enter the limits of the index:")
   (inputStrings
    ("\newline From:" " 10 \first\ EM)
    ("\newline To:\space{2} 10 \last\ EM))
    (doneButton "Continue" bcProductGen))))
  (htShowPage))

defun bcProductGen

(defun bcProductGen (htPage)
  (let (mand index car last)
    (setq mand ((htpLabelInputString htPage '|mand|))
    (setq index ((htpLabelInputString htPage '|index|))
    (setq car ((htpLabelInputString htPage '|first|))
    (setq last ((htpLabelInputString htPage '|last|))
    (bcGen (strconc "product(" mand "," index "," car "," last ")"))))

—— defun bcProductGen ——
defun Differentiate Basic Command

(htInitPage p1415)
(htMakePage p1416)
(htMakeDoneButton p1437)
(htShowPage p1416)
[$EmptyMode p640]

Pressing the Continue button will call the function bcDifferentiateGen due to this line:

(|htMakeDoneButton| "Continue" |bcDifferentiateGen|)

— defun bcDifferentiate —

(defun |bcDifferentiate| ()
(declare (special |$EmptyMode|))
(htInitPage| '|Differentiate Basic Command| nil)
(htMakePage)
`
((|domainConditions| (|isDomain| EM |$EmptyMode|)
 (|isDomain| S (|String|)) (|isDomain| SY (|Symbol|)))
 (|text| . "\newline ")
 (|text| . "\menuitemstyle{}\tab{2}"
 (|text| . "\newline List the {\em function} you want to differentiate:"))
 (|text| . "\newline\tab{2}"
 (|bcStrings| (55 "sin(x*y)" |diffand| EM))
 (|text| . "\blankline") (|text| . "\newline ")
 (|text| . "\menuitemstyle{}\tab{2}"
 (|text| . "\newline List the {\em variables} you want to differentiate with respect to?"
 (|text| . "\newline\tab{2}"
 (|bcStrings| (55 "x y" |variables| S . |quoteString|))
 (|text| . "\blankline") (|text| . "\newline ")

Enter the function you want to differentiate:

\sin(x*y)

List the variables you want to differentiate with respect to?

x y

List the number of times you want to differentiate with respect to each variable (leave blank if once for each)

1 2

Continue
CHAPTER 42. HYPERDOC BASIC COMMAND SUPPORT

- \text{List the number of times you want to differentiate with respect to each variable (leave blank if once for each)}

\text{ times}  
\text{quoteString}))))

\text{ভর্তির শেষ}

---

\text{defun bcDifferentiateGen}

\text{htpLabelInputString p1409}
\text{bcString2WordList p1400}
\text{bcwords2liststring p1400}
\text{length p??]
\text{bcError p1401]
\text{strconc p??]\n\text{bcGen p1399]

--- defun bcDifferentiateGen ---

\text{diffand}
\text{variables}
\text{times}]
\text{bcwords2liststring} varlist
\text{varpart}]
\text{indexpart}}
\text{bcError]
\text{strconc}]
\text{bcGen]

You must say how many times you want to differentiate with respect to each variable---or leave

(setq lastPart (if indexpart (strconc "," indexpart ")") "")

(strconc "differentiate(" mand "," varpart lastPart)))

---
defun Draw Basic Command

[htInitPage p1415]
[bcHt p1414]
[htShowPage p1416]

What would you like to draw?

Two Dimensional Plots

A function of one variable  \( y = f(x) \)
A parametrically defined curve  \((x(t), y(t))\)
A solution to a polynomial equation  \( p(x,y) = 0 \)

Three Dimensional Surfaces

A function of two variables  \( z = f(x,y) \)
A parametrically defined tube  \((x(t), y(t), z(t))\)
A parameterically defined surface  \((x(u,v), y(u,v), z(u,v))\)

— defun bcDraw —
defun Draw Basic Command by Function

Pressing the Continue button will call the function bcDraw2DfunGen due to this line:

```
(defun bcDraw2Dfun ()
  (declare (special $EmptyMode))

  (htInitPage "Draw Basic Command" NIL)
  (htMakePage
   '(((domainConditions
     (isDomain F (Float)) (isDomain SY (Symbol)))
     (text "\centerline{Drawing {\em y = f(x)}}\newline "
           \centerline{where {\em y} is the dependent variable and}\newline "
           \centerline{where {\em x} is the independent variable}\vspace{1}\newline "
           \menuitemstyle{}\tab{2}What {\em function} f would you like to draw?\newline\tab{2}"
     )
     )
     .
     (bcStrings (55 "x*cos(x)" |function| EM))
     (text .
      "\vspace{1}\newline\menuitemstyle{}\tab{2}Enter {\em dependent} variable:"
     )
     (bcStrings (6 |y| |dependent| SY)))

  )

```
42.2. FUNCTIONS CREATING PAGES

---

defun bcDraw2DfunGen

(let (fun dep ind from1 to1 title titlePart)
  (setq fun (htpLabelInputString htPage '|function|))
  (setq dep (htpLabelInputString htPage '|dependent|))
  (setq ind (htpLabelInputString htPage '|ind|))
  (setq from1 (htpLabelInputString htPage '|from1|))
  (setq to1 (htpLabelInputString htPage '|to1|))
  (setq title (htpLabelInputString htPage '|title|))
  (cond
    ((not (string-equal title ""))
     (setq titlePart (strconc "{} " title " ")
     (bcFinish "draw" fun (bcDrawIt2 ind from1 to1) titlePart))
    (t
     (bcFinish "draw" fun (bcDrawIt2 ind from1 to1)))))

---

defun Draw Basic Command by Parameters

[htInitPage p1415]
[htMakePage p1416]
CHAPTER 42. HYPERDOC BASIC COMMAND SUPPORT

Pressing the Continue button will call the function \texttt{bcDraw2DparGen} due to this line:

\begin{verbatim}
\texttt{(doneButton \texttt{"Continue" \texttt{bcDraw2DparGen}})}
\end{verbatim}

— \texttt{defun bcDraw2Dpar} —

\begin{verbatim}
(defun \texttt{bcDraw2Dpar} ()
  (declare (special \$EmptyMode)))
  (htInitPage \"Draw Basic Command\" NIL)
  (htMakePage
   '(((domainConditions (isDomain EM \$EmptyMode))
     (isDomain F (Float)) (isDomain SY (Symbol)))
     (text \"Drawing a parametrically defined curve:\)
     (text \"( f1(t), f2(t) )\)
     (text \"in terms of two functions f1 and f2\)
     (text \"and an independent variable t\)
     (text \"\vspace{1}\newline\textmenustyle{}\tab{2}Enter the two \emph{functions:}\)
     (bcStrings (44 \"-9*sin(4*t/5)\" \texttt{function1} EM))
     (bcStrings (44 \"8*sin(t)\" \texttt{function2} EM))
     (text \"\vspace{1}\newline\textmenustyle{}\tab{2}Enter \emph{independent variable and range:}\)
     (text \"\textmenustyle{}\tab{2}{\emph{Variable:}} \texttt{t\texttt{ind}} \textmenustyle{}\text{ranges from:} \texttt{-5*\pi} \text{ to:} \texttt{5*\pi})
     (text \"\vspace{1}\newline\textmenustyle{}\tab{2}Optionally enter a title for your curve: \texttt{Lissajous})
     (text \"\vspace{1}\newline\textmenustyle{}\tab{2})
     \texttt{Continue}
   )
\end{verbatim}
42.2. FUNCTIONS CREATING PAGES

(defun bcDraw2DparGen
   (htPage)
   (let (fun1 fun2 ind from1 to1 title curvePart titlePart)
      (setq fun1 (httpLabelInputString htPage ’|function1|))
      (setq fun2 (httpLabelInputString htPage ’|function2|))
      (setq ind (httpLabelInputString htPage ’|ind|))
      (setq from1 (httpLabelInputString htPage ’|from1|))
      (setq to1 (httpLabelInputString htPage ’|to1|))
      (setq title (httpLabelInputString htPage ’|title|))
      (setq curvePart (strconc "curve(" {} "\)\) " fun1 ",{}" fun2 ")")
      (cond
         ((not (string-equal title "))
            (setq titlePart (strconc "]" "title =="" title "]"))
            (bcFinish "draw" curvePart (bcDrawIt2 ind from1 to1) titlePart))
         (t
            (bcFinish "draw" curvePart (bcDrawIt2 ind from1 to1)))))

---

defun Draw Basic Command by Equation Solution

(htInitPage)
(htMakePage)
(htMakeDoneButton)
(htShowPage)
Pressing the Continue button will call the function bcDraw2DSolveGen due to this line:

```
(l|htMakeDoneButton| "Continue" 'bcDraw2DSolveGen)
```

---

```lisp
(defun bcDraw2DSolve ()
  (declare (special $EmptyMode))
  (|htInitPage| "Draw Basic Command" nil)
  (|htMakePage|
    '((|domainConditions| (|isDomain| EM $EmptyMode) (|isDomain| F (|Float|)) (|isDomain| SY (|Symbol|)))
      (|text| "Plotting the solution to \( p(x,y) = 0 \), where \( p \) is a polynomial in two variables \( x \) and \( y \)."
       "\( \text{Enter the polynomial} p: \)
       "\( y^2+7x^2y-(x^3+16x) \)"
      (|text| "\( \text{Enter the variables:} \)
       "\( x \) \text{ranges from:} -15 \text{ to:} 10"
       "\( y \) \text{ranges from:} -10 \text{ to:} 50"
    (|text| "\text{Optionally enter a title for your curve:}"
   }
   (|bcStrings| (40 "y**2+7*x**2*y-(x**3+16*x)" |function| EM))
  (|text| "\( \text{Enter the variables:} \)
   "\( x \) \text{ranges from:} -15 \text{ to:} 10"
   "\( y \) \text{ranges from:} -10 \text{ to:} 50"
   "\text{Optionally enter a title for your curve:}"
  }
  (|bcStrings| (40 "y**2+7*x**2*y-(x**3+16*x)" |function| EM)))
```

---

```lisp
(defun bcDraw2DSolve ()
  (declare (special $EmptyMode))
  (|htInitPage| "Draw Basic Command" nil)
  (|htMakePage|
    '((|domainConditions| (|isDomain| EM $EmptyMode) (|isDomain| F (|Float|)) (|isDomain| SY (|Symbol|)))
      (|text| "Plotting the solution to \( p(x,y) = 0 \), where \( p \) is a polynomial in two variables \( x \) and \( y \)."
       "\( \text{Enter the polynomial} p: \)
       "\( y^2+7x^2y-(x^3+16x) \)"
      (|text| "\( \text{Enter the variables:} \)
       "\( x \) \text{ranges from:} -15 \text{ to:} 10"
       "\( y \) \text{ranges from:} -10 \text{ to:} 50"
    (|text| "\text{Optionally enter a title for your curve:}"
   }
   (|bcStrings| (40 "y**2+7*x**2*y-(x**3+16*x)" |function| EM))
  (|text| "\( \text{Enter the variables:} \)
   "\( x \) \text{ranges from:} -15 \text{ to:} 10"
   "\( y \) \text{ranges from:} -10 \text{ to:} 50"
   "\text{Optionally enter a title for your curve:}"
  }
  (|bcStrings| (40 "y**2+7*x**2*y-(x**3+16*x)" |function| EM)))
```
42.2. FUNCTIONS CREATING PAGES
(|bcStrings| (9 50 |to2| F))
(|text| "\\indent{0}\\vspace{1}\\newline\\menuitemstyle{}\\tab{2} "
"Optionally enter a {\\em title} for your curve:")
(|bcStrings| (15 "" |title| S)) (|text| . "\\indent{0}")))
(|htMakeDoneButton| "Continue" ’|bcDraw2DSolveGen|)
(|htShowPage|))

———-

defun bcDraw2DSolveGen
[htpLabelInputString p1409]
[strconc p??]
[bcFinish p1397]
— defun bcDraw2DSolveGen —
(defun |bcDraw2DSolveGen| (htPage)
(let (fun ind1 from1 to1 ind2 from2 to2 title clipPart titlePart)
(setq fun
(|htpLabelInputString| htPage ’|function|))
(setq ind1 (|htpLabelInputString| htPage ’|independent1|))
(setq from1 (|htpLabelInputString| htPage ’|from1|))
(setq to1
(|htpLabelInputString| htPage ’|to1|))
(setq ind2 (|htpLabelInputString| htPage ’|independent2|))
(setq from2 (|htpLabelInputString| htPage ’|from2|))
(setq to2
(|htpLabelInputString| htPage ’|to2|))
(setq title (|htpLabelInputString| htPage ’|title|))
(setq clipPart (strconc "{}" "range==[{}"
from1 ".." to1 ’|,{}|
from2 ".." to2 "]"))
(cond
((not (string-equal title ""))
(setq titlePart (strconc "{}" "title ==\"" title "\""))
(|bcFinish| "draw" (strconc fun " = 0 ") ind1 ind2 clipPart titlePart))
(t
(|bcFinish| "draw" (strconc fun " = 0 ") ind1 ind2 clipPart)))))

———-

defun Draw Basic Command by 3D function
[htInitPage p1415]
[htMakePage p1416]
[htShowPage p1416]

1357


CHAPTER 42. HYPERDOC BASIC COMMAND SUPPORT

Pressing the Continue button will call the function $\texttt{bcDraw3DfunGen}$ due to this line:

$$\text{(|doneButton| "Continue" |bcDraw3DfunGen|)}$$

--- defun bcDraw3Dfun ---

(defun |bcDraw3Dfun| ()
(declare (special |$\texttt{EmptyMode}$|))
(htInitPage "Three Dimensional Draw Basic Command" nil)
(htMakePage
 '(((|domainConditions| (|isDomain| EM |$\texttt{EmptyMode}$|))
  (|isDomain| F (|Float|)) (|isDomain| SY (|Symbol|)))
  (|text| "\text{\centerline{Drawing $z = f(x,y)$}}\newline \
\text{\centerline{where $z$ is the dependent variable and}}\newline \
\text{\centerline{where $x, y$ are the independent variables}}\vspace{1}\newline\itemstyle{}\tab{2} \
\text{\centerline{What function $f$ which you like to draw?}}\newline\itemstyle{}\tab{2}
$\exp(\cos(x-y) - \sin(x*y)) - 2$\newline \
|Enter dependent variable:| $z$
\newline\itemstyle{}\tab{2} \
\text{\centerline{Enter independent variables and ranges:}}\newline\itemstyle{}\tab{2} \
\text{\centerline{Variable: $x$ ranges from:} $-5$ to: $5$}\newline \
\text{\centerline{Variable: $y$ ranges from:} $-5$ to: $5$}\newline \
\text{\centerline{Optionally enter a title for your surface:}}\newline\itemstyle{}\tab{2} \
\text{(Continue)\newline \
\text{\centerline{What function} $f$ which you like to draw?}$\newln\tab{2}$\newline \
|bcStrings| (55 "$\exp(\cos(x-y) - \sin(x*y)) - 2$ |function| EM))\newline \
|text| . \
$\newln\itemstyle{}\tab{2}$\text{\centerline{Enter dependent} variable:"}\newline \
|bcStrings| (6 |z| |dependent| SY))\newline \
|text| . "$\newln\itemstyle{}\tab{2}$\text{\centerline{Enter} \text{\centerline{variables and ranges:}}$\newln\tab{2}$" \
|bcStrings| (6 |x| |independent1| SY))\newline \
|text| . "$\newln\itemstyle{}\tab{2}$\text{\centerline{ranges} \text{\centerline{from:}}$\newln\tab{2}$" \
|bcStrings| (9 -5 |from1| F))\newline \
|text| . "$\newln\itemstyle{}\tab{2}$"

Pressing the Continue button will call the function $\texttt{bcDraw3DfunGen}$ due to this line:
42.2. FUNCTIONS CREATING PAGES

(|bcStrings| (9 5 |to1| F))
(|text| . "\newline\tab{2}{\em Variable:}")
(|bcStrings| (6 |y| |independent2| SY))
(|text| . "\em from:\")
(|bcStrings| (9 -5 |from2| F)) (|text| . "\em to:\")
(|bcStrings| (9 5 |to2| F))
(|text| "\indent{0}\vspace{1}\newline\menuitemstyle{}\tab{2} "
  "Optionally enter a \em title for your surface:\")
(|bcStrings| (15 "" |title| S)) (|text| . "\indent{0}"")
(|doneButton| "Continue" |bcDraw3DfunGen|))
(|htShowPage|))

__________________________

defun bcDraw3DfunGen

[htpLabelInputString p1409]
[stconc p??]
[bcFinish p1397]
[bcDrawIt2 p1342]

— defun bcDraw3DfunGen —

(defun |bcDraw3DfunGen| (htPage)
  (let (fun dep ind1 from1 to1 ind2 from2 to2 title titlePart)
    (setq fun (|htpLabelInputString| htPage '|function|))
    (setq dep (|htpLabelInputString| htPage '|dependent|))
    (setq ind1 (|htpLabelInputString| htPage '|independent1|))
    (setq from1 (|htpLabelInputString| htPage '|from1|))
    (setq to1 (|htpLabelInputString| htPage '|to1|))
    (setq ind2 (|htpLabelInputString| htPage '|independent2|))
    (setq from2 (|htpLabelInputString| htPage '|from2|))
    (setq to2 (|htpLabelInputString| htPage '|to2|))
    (setq title (|htpLabelInputString| htPage '|title|))
    (cond
      ((not (string-equal title ""))
       (setq titlePart (stconc "{} "title ==\"" title \"")))
      (bcFinish| "draw" fun
       (|bcDrawIt2| ind1 from1 to1)
       (|bcDrawIt2| ind2 from2 to2) titlePart))
    (t
      (bcFinish| "draw" fun
       (|bcDrawIt2| ind1 from1 to1)
       (|bcDrawIt2| ind2 from2 to2))))

__________________________
defun Draw Basic Command by 3D parameterized tube

[htInitPage p1415]
[htMakePage p1416]
[htShowPage p1416]
[$EmptyMode p640]

```lisp
(defun bcDraw3DparGen ()
  (declare (special $EmptyMode))
  (|htInitPage| "Draw Basic Command" NIL)
  (|htMakePage|
    '((|domainConditions| (|isDomain| EM $EmptyMode) (|isDomain| F (|Float|)) (|isDomain| SY (|Symbol|)))
      (|text| "\centerline{Drawing a parametrically defined curve: ( f1(t), f2(t), f3(t) )}"
        in terms of three functions f1, f2, and f3 and an independent variable t

    Enter the three functions of the independent variable:
    Function f1: 1.3*cos(2*t)*cos(4*t) + sin(4*t)*cos(t)
    Function f2: 1.3*sin(2*t)*cos(4*t) - sin(4*t)*sin(t)
    Function f3: 2.5*cos(4*t)

    Enter independent variable and range:
    Variable: t ranges from: 0 to: 4*%pi

    Optionally enter a title for your surface: knot

    Continue

Pressing the Continue button will call the function bcDraw3DparGen due to this line:

(|doneButton| "Continue" |bcDraw3DparGen|)

---

(defun bcDraw3Dpar ()
  (declare (special $EmptyMode))
  (|htInitPage| "Draw Basic Command" NIL)
  (|htMakePage|
    '((|domainConditions| (|isDomain| EM $EmptyMode))
      (|isDomain| F (|Float|)) (|isDomain| SY (|Symbol|)))
    (|text| "\centerline{Drawing a parametrically defined curve:}"
      "\{\em ( f1(t), f2(t), f3(t) )}\}\\newline"
    "\centerline{in terms of three functions f1, f2, and f3}\}\\newline"
    "\centerline{and an independent variable t}\}\\newline\\vspace{1}\\newline\\menuitemstyle{}\tab{2}\centerline{Enter the three f}"
      "\centerline{unctions of the independent variable:}\}"
    (|text| . "\{\newline\tab{2}\{\em Function f1:}\}"
      (|bcStrings| (42 "1.3*cos(2*t)*cos(4*t) + sin(4*t)*cos(t)" |function1| EM))
    (|text| . "\{\newline\tab{2}\{\em Function f2:}\)"
42.2. FUNCTIONS CREATING PAGES

\textbf{defun bcDraw3DparGen}

\texttt{[htpLabelInputString p1409]}

\texttt{[strconc p??]}

\texttt{[bcFinish p1397]}

\texttt{[bcDrawIt2 p1342]}

\begin{verbatim}
(defun |bcDraw3DparGen| (htPage)
  (let ((fun1 fun2 fun3 ind from1 to1 title curvePart tubePart titlePart)
    (setq fun1 (|htpLabelInputString| htPage '|function1|))
    (setq fun2 (|htpLabelInputString| htPage '|function2|))
    (setq fun3 (|htpLabelInputString| htPage '|function3|))
    (setq ind (|htpLabelInputString| htPage '|ind|))
    (setq from1 (|htpLabelInputString| htPage '|from1|))
    (setq to1 (|htpLabelInputString| htPage '|to1|))
    (setq title (|htpLabelInputString| htPage '|title|))
    (setq curvePart (strconc "curve( {}" fun1 ",{}" fun2 ",{}" fun3 ")")
    (setq tubePart "{}tubeRadius==.25,{}tubePoints==16")
    (cond
      ((not (string-equal title ")))
        (setq titlePart (strconc "{}" title ")
        (|bcFinish| "draw" curvePart
          (|bcDrawIt2| ind from1 to1) tubePart titlePart))
      (t
        (|bcFinish| "draw" curvePart
          (|bcDrawIt2| ind from1 to1) tubePart))))
\end{verbatim}
defun Draw Basic Command by 3D parameterized function

[htInitPage p1415]
[htMakePage p1416]
[htMakeDoneButton p1437]
[htShowPage p1416]
[$EmptyMode p640]

Pressing the Continue button will call the function bcDraw3Dpar1Gen due to this line:

```
(\|htMakeDoneButton| "Continue" \'|bcDraw3Dpar1Gen|)
```

---

---

(defun |bcDraw3Dpar1| ()
 (declare (special |$EmptyMode|))
 (\|htInitPage| "Draw Basic Command" NIL)
 (\|htMakePage|)
 '(((\|domainConditions|) (\|isDomain| EM |$EmptyMode|))
   (\|isDomain| F (\|Float|)) (\|isDomain| SY (\|Symbol|)))
 (\|text| "\centerline{Drawing a parametrically defined surface:}\newline "
   "\centerline{{{\|em| ( f1(u,v), f2(u,v), f3(u,v) )}}}\newline "
   "\centerline{in terms of three functions f1, f2, and f3}\newline "
   "\centerline{and two independent variables u and v}\vspace{1}\newline\menuitemstyle{}\tab{2}"
 (\|text| . "Enter the three \{\|em| functions\} of the independent variables:"))
 (\|text| . "\newline\tab{2}"
 (\|text| . "{{\|em| Function f1:}}")
 (\|bcStrings| (43 "u*sin(v)" |function1| EM)))
\section*{42.2. \textit{FUNCTIONS CREATING PAGES}}

\begin{verbatim}
(defun bcDraw3Dpar1Gen (htPage)
  (let ((fun1 fun2 fun3 ind1 from1 to1 ind2 from2 to2
         title r1 r2 surfacePart titlePart)
        (setq fun1 (\texttt{|htpLabelInputString| htPage \texttt{"|function1|}}))
        (setq fun2 (\texttt{|htpLabelInputString| htPage \texttt{"|function2|}}))
        (setq fun3 (\texttt{|htpLabelInputString| htPage \texttt{"|function3|}}))
        (setq ind1 (\texttt{|htpLabelInputString| htPage \texttt{"|ind1|}}))
        (setq from1 (\texttt{|htpLabelInputString| htPage \texttt{"|from1|}}))
        (setq to1 (\texttt{|htpLabelInputString| htPage \texttt{"|to1|"}}))
        (setq from2 (\texttt{|htpLabelInputString| htPage \texttt{"|from2|"}}))
        (setq to2 (\texttt{|htpLabelInputString| htPage \texttt{"|to2|"}}))

        (bcDrawIt2 htPage)

        (bcFinish htPage)
      )
)
\end{verbatim}

\begin{itemize}
\item \texttt{Optionally enter a \textit{title} for your surface:}"
\end{itemize}

\begin{verbatim}
"\indent{0}"\newline\menuitemstyle{}\tab{2} \\
\indent{0}"
\end{verbatim}

\begin{verbatim}
defun bcDraw3Dpar1Gen
[htpLabelInputString p1409]
[bcDrawIt2 p1342]
[strconc p??]
[bcFinish p1397]

— defun bcDraw3Dpar1Gen —
	
\end{verbatim}
defun Series Basic Command

(htInitPage p1415)
(htMakePage p1416)
(htShowPage p1416)
[$EmptyMode p640]

Create a series by:
- Expansion  Expand a function in a series around a point
- Formula    Give a formula for the i'th coefficient

— defun bcSeries —

(defun bcSeries ()
  (declare (special $EmptyMode))
  (htInitPage "Series Basic Command" nil)
  (htMakePage
    '((domainConditions
        (isDomain EM $EmptyMode)
          (isDomain S (String))
          (isDomain SY (Symbol)))
      (text . "Create a series by:")
      (text . "\beginmenu")
      (text . "\item")
      (bcLinks ("\menuitemstyle{Expansion}" " bcSeriesExpansion" nil))
      (text . "\item")
      (text . \"\tab\{i\} Expand a function in a series around a point")
      (text . "\item")
      (bcLinks ("\menuitemstyle{Formula}" " bcSeriesByFormula" nil))
      (text . "\item")
      (text . \"\tab\{i\} Give a formula for the {{em i}}'th coefficient")
      (text . \"\endmenu")
    )
  )
  (htShowPage))
defun Series Basic Command expand around a point

[htInitPage p1415]
[htMakePage p1416]
[htMakeDoneButton p1437]
[htShowPage p1416]
[$EmptyMode p640]

Pressing the Continue button will call the function bcSeriesExpansionGen due to this line:

\[(|htMakeDoneButton| "Continue" '|bcSeriesExpansionGen|)\]

— defun bcSeriesExpansion —

(defun |bcSeriesExpansion| (a b)
  (declare (ignore a b))
  (declare (special |$EmptyMode|))
  (|htInitPage| "Series Expansion Basic Command" nil)
  (|htMakePage|)
  '((|domainConditions| (|isDomain| EM |$EmptyMode|)
    (|isDomain| EEM (|Expression| |$EmptyMode|))
    (|isDomain| S (|String|)) (|isDomain| SY (|Symbol|)))
    (|text| . "\newline ")
    (|text| . "\menuitemstyle{}\tab{2}"
    (|text| . "Enter the {\em function} you want to expand in a power series"
    (|text| . "\newline\tab{2} ")
    (|bcStrings| (55 "log(cot(x))" |function| EM))
    (|text| . "\blankline ")
    (|text| . "\newline ")
    (|text| . "\menuitemstyle{}\tab{2}"
    (|text| . "Enter the {\em power series variable}"
    (|text| . "\tab{49}"
    (|bcStrings| (8 |x| |variable| SY))
    (|text| . "\blankline ")
    (|text| . "\newline ")
defun bcSeriesExpansionGen

(defun bcSeriesExpansionGen (htPage)
  (let (fun var point terms)
    (setq fun (|htpLabelInputString| htPage '|function|))
    (setq var (|htpLabelInputString| htPage '|variable|))
    (setq point (|htpLabelInputString| htPage '|point|))
    (setq terms (|htpLabelInputString| htPage '|numberOfTerms|))
    (|bcFinish| "series" fun (strconc var " = 
                      point))))

defun Series Basic Command series by formula

(defun bcSeriesByFormula |htInitPage p1415|
  |htMakePage p1416|
  |htShowPage p1416|

  Select the kind of power series you want to create:
  ☑ Taylor Series
    Series where the exponent ranges over the integers from a non-negative integer value to plus infinity by an arbitrary positive integer step size
  ☑ Laurent Series
    Series where the exponent ranges from an arbitrary integer value to plus infinity by an arbitrary positive integer step size
  ☑ Puiseux Series
    Series where the exponent ranges from an arbitrary rational value to plus infinity by an arbitrary positive rational number step size

— defun bcSeriesByFormula —
(defun |bcSeriesByFormula| (a b)
  (declare (ignore a b))
  (|htInitPage| "Power Series Basic Command" NIL)
  (|htMakePage| '((|text| . "Select the kind of power series you want to create:")
    (|text| . "\beginmenu") (|text| . "\item")
    (|bcLinks| "\menuitemstyle{Taylor Series}" "|bcTaylorSeries| |taylor|)
    (|text| . "Series where the exponent ranges over the integers from a non-negative integer value to plus infinity by an arbitrary positive integer step size")
    (|text| . "\item")
    (|bcLinks| "\menuitemstyle{Laurent Series}" "|bcLaurentSeries| |laurent|)
    (|text| . "Series where the exponent ranges from an arbitrary integer value to plus infinity by an arbitrary positive integer step size")
    (|text| . "\item")
    (|bcLinks| "\menuitemstyle{Puiseux Series}" "|bcPuiseuxSeries| |puiseux|)
    (|text| . "Series where the exponent ranges from an arbitrary rational value to plus infinity by an arbitrary positive rational number step size")
    (|text| . "\endmenu")))
  (|htShowPage|))

defun Taylor Series Basic Command

[htInitPage p1415]
[htMakePage p1416]
[htShowPage p1416]
[$EmptyMode p640]
Pressing the **Continue** button will call the function `bcTaylorSeriesGen` due to this line:

```lisp
(|doneButton| "Continue" |bcTaylorSeriesGen|)
```

---

**defun bcTaylorSeries**

```lisp
(defun bcTaylorSeries (a b)
  (declare (ignore a b))
  (htInitPage| "Taylor Series Basic Command" NIL)
  (htMakePage
   '#( (|domainConditions| (|isDomain| EM |$EmptyMode|)
       (|isDomain| EEM (|Expression| |$EmptyMode|))
       (|isDomain| S (|String|)) (|isDomain| SY (|Symbol|)))
   (|text| . "Enter the formula for the general coefficient of the series")
   (|text| . "Enter the \emph{index variable} for your formula")
   (|text| . "Enter the \emph{power series variable}"
     (|text| . "Enter the \emph{point} about which you want to expand")
   (|text| . "Enter the \emph{initial value of the index (an integer)}")
   (|text| . "Enter the \emph{step size (a positive integer)}")
  (|bcStrings| (55 "1/factorial(i)" |formula| EM))
  (|text| . "Enter the \emph{index variable} for your formula")
  (|text| . "Enter the \emph{power series variable}"
    (|text| . "Enter the \emph{point} about which you want to expand")
  (|text| . "Enter the \emph{initial value of the index (an integer)}")
  (|text| . "Enter the \emph{step size (a positive integer)}")
  )
```

For Taylor Series, the exponent of the power series variable ranges from an initial value, an arbitrary non-negative integer, to plus infinity; the step size is any positive integer.
42.2. FUNCTIONS CREATING PAGES

"For Taylor Series, the exponent of the power series variable ranges from an \emph{initial value}, an arbitrary non-negative integer, to plus infinity; the \emph{step size} is any positive integer."

- Enter the \emph{initial value} of the index (an integer)
- Enter the \emph{step size} (a positive integer)

---

defun bcSeriesByFormulaGen

---

defun Laurent Series Basic Command

---
CHAPTER 42. HYPERDOC BASIC COMMAND SUPPORT

Pressing the *Continue* button will call the function `bcLaurentSeriesGen` due to this line:

```lisp
|doneButton| "Continue" |bcLaurentSeriesGen|
```

---

**defun bcLaurentSeries**

```lisp
(defun bcLaurentSeries (a b)
  (declare (special |$EmptyMode|) (ignore a b))
  (|htInitPage| "Laurent Series Basic Command" NIL)
  (|htMakePage|
   '(((|domainConditions| (|isDomain| EM |$EmptyMode|)
       (|isDomain| EEM (|Expression| |$EmptyMode|))
       (|isDomain| S (|String|)) (|isDomain| I (|Integer|))
       (|isDomain| PI (|PositiveInteger|))
       (|isDomain| SY (|Symbol|))))
   ([text] . "\newline")
   ([text] . "\menuitemstyle{}\tab{2}"
     (|text| . "Enter the formula for the general coefficient of the series")
     (|text| . "\newline\tab{2}"
       (|bcStrings| (55 "(-1)**(n - 1)/(n + 2)" |formula| EM)))
   ([text] . "\vspace{1}\newline"
     (|text| . "\menuitemstyle{}\tab{2}"
       (|text| . "Enter the {\em index variable} for your formula")
       (|text| . "\tab{49}"
         (|bcStrings| (8 |n| |index| SY)))
     (|text| . "\newline"
       (|text| . "\menuitemstyle{}\tab{2}"
         (|text| . "Enter the {\em power series variable}"
           (|text| . "\tab{49}"
             (|bcStrings| (8 |x| |variable| SY))))
     (|text| . "\newline"
       (|text| . "\menuitemstyle{}\tab{2}"
         (|text| . "Enter the {\em point} about which you want to expand"
           (|text| . "\tab{49}"
             (|bcStrings| (8 0 |point| F)))))
```

For Laurent series, the exponent of the power series variable ranges from an initial value, an arbitrary integer value, to plus infinity; the step size is any positive integer.
42.2. FUNCTIONS CREATING PAGES

```
(defun Puiseux Series Basic Command
    (defun |bcPuiseuxSeries| (a b)
        (declare (special |$EmptyMode|) (ignore a b))
```
CHAPTER 42. HYPERDOC BASIC COMMAND SUPPORT

(\htInitPage "Puiseux Series Basic Command" nil)
(\htMakePage
  '(\domainConditions
     (\isDomain EM ($EmptyMode))
     (\isDomain EEM (\Expression $EmptyMode))
     (\isDomain S (\String))
     (\isDomain I (\Integer))
     (\isDomain PI (\PositiveInteger))
     (\isDomain RN (\Fraction (\Integer)))
     (\isDomain SY (\Symbol))))
(\text . "\newline"
(\text . "\menuitemstyle{}\tab{2}"
(\text . "Enter the {\em formula} for the general coefficient of the series"
(\text . "\newline"
(\bcStrings (55 "(-1)**((3*n - 4)/6)/factorial(n - 1/3)" |formula| EM))
(\text . "\vspace{1}\newline"
(\text . "\menuitemstyle{}\tab{2}"
(\text . "Enter the {\em index variable} for your formula"
(\text . "\tab{49}"
(\bcStrings (8 |n| |index| SY))
(\text . "\newline"
(\text . "\menuitemstyle{}\tab{2}"
(\text . "Enter the {\em power series variable}"
(\text . "\tab{49}"
(\bcStrings (8 |x| |variable| SY))
(\text . "\newline"
(\text . "\menuitemstyle{}\tab{2}"
(\text . "Enter the {\em point} about which you want to expand"
(\text . "\tab{49}"
(\bcStrings (8 0 |point| F))
(\text . "\newline"
(\text . "\menuitemstyle{}\tab{2}"
(\text . "Enter the {\em initial value} of index (a rational number)"
(\text . "\tab{51}"
(\bcStrings (6 "4/3" |min| RN))
(\text . "\newline"
(\text . "\menuitemstyle{}\tab{2}"
(\text . "Enter the {\em step size} (a positive rational number)"
(\text . "\tab{51}"
(\bcStrings (6 "2" |step| RN))
(\doneButton "Continue" |bcPuiseuxSeriesGen|)))
(\htShowPage))

defun bcTaylorSeriesGen
[bcSeriesGen p1373]

— defun bcTaylorSeriesGen —
(defun |bcTaylorSeriesGen| (htPage)
  (|bcSeriesGen| htPage))

---

defun bcLaurentSeriesGen

[bcSeriesGen p1373]

— defun bcLaurentSeriesGen —

(defun |bcLaurentSeriesGen| (htPage)
  (|bcSeriesGen| htPage))

---

defun bcPuiseuxSeriesGen

[bcSeriesGen p1373]

— defun bcPuiseuxSeriesGen —

(defun |bcPuiseuxSeriesGen| (htPage)
  (|bcSeriesGen| htPage))

---

defun bcSeriesGen

[htpLabelInputString p1409]
[stconc p??]
[bcFinish p1397]

— defun bcSeriesGen —

(defun |bcSeriesGen| (htPage)
  (let (step min formula index var point varPart minPart)
    (setq step (|htpLabelInputString| htPage '|step|))
    (setq min (|htpLabelInputString| htPage '|min|))
    (setq formula (|htpLabelInputString| htPage '|formula|))
    (setq index (|htpLabelInputString| htPage '|index|))
    (setq var (|htpLabelInputString| htPage '|variable|))
defun Limit Basic Command

\beginquote
What kind of limit do you want to compute?

\beginmenu
\item \text{A real limit?} The limit as the variable approaches a real value along the real axis
\item \text{A complex limit?} The limit as the variable approaches a complex value along any path in the complex plane
\endmenu
\endquote

(defun bcLimit ()
  (declare (special $EmptyMode$))
  (htInitPage "Limit Basic Command" NIL)
  (htMakePage
   '((domainConditions (isDomain EM $EmptyMode$)
     (isDomain S (|String|)) (isDomain SY (|Symbol|)))
     (text . "What kind of limit do you want to compute? ")
     (text . "\blankline")
     (text . "\beginmenu")
     (text . "\item")
     (bcLinks (\menuitemstyle{A real limit?} "" |bcRealLimit| |real|))
     (text . "\indentrel{17}\tab{0}"
     (text . "The limit as the variable approaches a \{\em real\} value along the real axis")
     (text . "\indentrel{-17}"
     (bcLinks ("\menuitemstyle{A complex limit?}" "" |bcComplexLimit| |complex|))
     (text . "\indentrel{17}\tab{0}"
     (text . "The limit as the variable approaches a \{\em complex\} value along any path in the complex plane")
     (text . "\indentrel{-17}"
     (text . "\endmenu")
   )
  )
  (htShowPage))
defun Real Limit Basic Command

[htInitPage p1415]
[htMakePage p1416]
[htShowPage p1416]
[$EmptyMode p640]

Pressing the Continue button will call the function bcRealLimitGen due to this line:

```
(|doneButton| "Continue" |bcRealLimitGen|)
```

---

— defun bcRealLimit —

```
(defun |bcRealLimit| (a b)
  (declare (special |$EmptyMode|) (ignore a b))
  (|htInitPage| "Real Limit Basic Command" NIL)
  (|htMakePage|
    '((|domainConditions| (|isDomain| EM |$EmptyMode|)
        (|isDomain| S (|String|)) (|isDomain| F (|Float|))
        (|isDomain| SY (|Symbol|)))
      (|text| . "\newline")
      (|text| . "\menuitemstyle{}\tab{2}"
        "Enter the \em function\ you want to compute the limit of:")
      (|text| . "\newline\tab{2}
        \menuitemstyle{}"
      (|bcStrings| (45 "x*sin(1/x)" |expression| EM))
      (|text| . "\blankline"
        "Enter the name of the \em variable":
      (|text| . "\newline"
        \menuitemstyle{}"
      (|bcStrings| "x" |variable| EM)
      (|text| . "\newline"
        "Compute the limit at"
      (|text| . "\tab{2}"
        "A finite point:
      (|text| . "\tab{41}
        "Plus infinity
      (|text| . "\tab{41}
        "Minus infinity
    )
  )
```

---
defun bcRealLimitGen (htPage)
(let (|p| |fun| |var| |loc| |page|)
  (cond
    ((not (eq (setq |p| (|htpButtonValue| htPage '|location|)) '|finitePoint|))
     (setq |fun| (|htpLabelInputString| htPage '|expression|))
     (setq |var| (|htpLabelInputString| htPage '|variable|))
     (setq |loc|
       (if (eq |p| '|plusInfinity|) "%plusInfinity" "%minusInfinity")))
    (|bcFinish| "limit" |fun| (strconc |var| " = " |loc|)))
(t
  (setq |page| (|htInitPage| "Real Limit Basic Command" nil))
  (|htMakePage|
   '((|text| . "Compute the limit")
     (|lispLinks|
       ("\menuitemstyle{From both directions}" "|bcRealLimitGen1| |both|)
42.2. FUNCTIONS CREATING PAGES

(defun bcRealLimitGen1
  (htPage key)
  (let (direction fun var loc varPart)
    (setq direction
      (cond ((eq key '|right|) "\"right\")
            ((eq key '|left|) "\"left\")
            (t nil)))
    (setq fun (htpProperty htPage '|fun|))
    (setq var (htpProperty htPage '|var|))
    (setq loc (htpProperty htPage '|loc|))
    (setq varPart (strconc var " = " loc))
    (bcFinish "limit" fun varPart direction)))

defun Complex Limit Basic Command

(defun Complex Limit Basic Command
  (htInitPage)
  (htMakePage)
  (htShowPage)
  ($EmptyMode)
Pressing the Continue button will call the function `bcComplexLimitGen` due to this line:

```
|doneButton| "Continue" |bcComplexLimitGen|
```

---

```
defun bcComplexLimit (a b)
  (declare (special |$EmptyMode|) (ignore a b))
  (htInitPage| "Complex Limit Basic Command" nil)
  (htMakePage|
    '((domainConditions| (isDomain| EM |$EmptyMode|)
      (isDomain| S (|String|)) (isDomain| F (|Float|))
      (isDomain| SY (|Symbol|)))
    (text| . "\newline ") (text| . \menuitemstyle{}\tab{2}"
    (text| . "Enter the {\em function} you want to compute the limit of:"))
    (text| . "\newline\space{0}Real part:\space{3}
    (bcStrings| (20 0 |real| F))
    (text| . "\newline Complex part:"))
    (bcStrings| (20 0 |complex| F)))
  (radioButtons| location|
    ("A finite point:"
    (text| . "\newline\space{0}Real part:\space{3}"
     (bcStrings| (20 0 |real| F))
    (text| . "\newline Complex part:"))
    (bcStrings| (20 0 |complex| F)))
  (finitePoint|)
```

---
defun bcComplexLimitGen

(let (fun var p real comp complexPart loc varPart)
  (setq fun (htpLabelInputString p '|expression|))
  (setq var (htpLabelInputString p '|variable|))
  (setq loc (htpButtonValue p '|location|))
  (cond
    ((eq loc '|finitePoint|)
     (setq real (htpLabelInputString p '|real|))
     (setq comp (htpLabelInputString p '|complex|)))
  (setq complexPart
    (cond
      ((string= comp "0") "")
      ((string= comp "1") "%i")
      (t (strconc comp "*%i")(t)))))
  (cond
    ((string= real "0") (if (string= complexPart "") '|0| complexPart))
    ((string= complexPart "") real)
    (t (strconc real " + " complexPart)))
  (t "%infinity")))
  (setq varPart (strconc var " = " loc))
  (bcFinish "complexLimit" fun varPart)))

defvar $systemType

— initvars —

(setq |$systemType| nil)
defvar $numberOfEquations

— initvars —
(defvar $numberOfEquations 0)


defvar $solutionMethod

— initvars —
(defvar $solutionMethod nil)

defun Solve Basic Command

[htInitPage p1415]
[htMakePage p1416]
[htShowPage p1416]
[$EmptyMode p640]

What do you want to solve?

- A System Of Linear Equations
- A System of Polynomial Equations
- A Single Polynomial Equation

— defun bcSolve —

(defun |bcSolve| ()
  (|htInitPage| "Solve Basic Command" nil)
  (|htMakePage|
    '((|text| : "What do you want to solve? ")
      (|text| : "\beginmenu") (|text| : "\item ")
      (|bcLinks|
        ("\menuitemstyle{A System Of Linear Equations}" "
          |bcLinearSolve| |linear|)
      ))
defun Linear Solve Basic Command

(devan bcLinearSolve (p nn)
 (declare (ignore p nn))
 (htInitPage "Basic Solve Command" NIL)
 (htMakePage)
 '(((text) . "How do you want to enter the equations?
 (text) . "\beginmenu"
 (text) . "\item"
 (text) . "\newline"
 (bcLinks)
 ("\menuitemstyle{Directly as equations}" "
 |bcLinearSolveEqns| |equations|)
 (text) . "\item"
 (text) . "\newline"
 (bcLinks)
 ("\menuitemstyle{In matrix form} AX = B, where A is a matrix of coefficients and B is a vector"
 |bcLinearSolveMatrix| |matrix|)
 (text) . "\indentrel{16}\tab{0}"
 (text) . "AX = B, where $AX = B$, where $A$ is a matrix of coefficients and $B$ is a vector"
 (text) . "\indentrel{-16}\item"
defun Linear Solve Equations Basic Command
[htInitPage p1415]
[htMakePage p1416]
[htMakeDoneButton p1437]
[htShowPage p1416]
[$EmptyMode p640]

Pressing the Continue button will call the function bcLinearSolveEqns1 due to this line:

((htMakeDoneButton "Continue" '|bcLinearSolveEqns1|)

— defun bcLinearSolveEqns —

(defun |bcLinearSolveEqns| (htPage p)
  (declare (ignore htPage p))
  ((htInitPage "Basic Solve Command" nil)
   (htMakePage
    '(((|domainConditions| ((|isDomain| PI (|PositiveInteger|)))
      (|inputStrings|
       ("Enter the \{\em number\} of equations:" " 5 2
        |numberOfEquations| PI))))
    (htMakeDoneButton "Continue" '|bcLinearSolveEqns1|)
    (htShowPage|)))

—

defun bcSystemSolve
[htInitPage p1415]
[htMakePage p1416]
[htMakeDoneButton p1437]
Pressing the Continue button will call the function \texttt{bcSystemSolveEqns1} due to this line:

\begin{verbatim}
(\texttt{|htMakeDoneButton| "Continue" \texttt{'}|bcSystemSolveEqns1|})
\end{verbatim}

---

\textbf{defun \texttt{bcSystemSolve} ---}

\begin{verbatim}
(defun \texttt{bcSystemSolve} (htPage p)
 (declare (ignore htPage p))
 (\texttt{|htInitPage| "Basic Solve Command" NIL)
 (\texttt{|htMakePage|}
 ('((\texttt{|domainConditions| (\texttt{|isDomain| PI \texttt{(\texttt{|PositiveInteger|)}}))
 (\texttt{|inputStrings|}
  ("Enter the \{\texttt{em} number\} of equations:" " 5 2
  \texttt{|numberOfEquations| PI)))
 (\texttt{|htMakeDoneButton| "Continue" \texttt{'}|bcSystemSolveEqns1|})
 (\texttt{|htShowPage|}))
\end{verbatim}

---

\textbf{defun \texttt{bcSolveSingle} ---}

\begin{verbatim}
(defun \texttt{bcSolveSingle} (htPage p)
 (declare (ignore p))
 (\texttt{|htpSetProperty| htPage \texttt{'}|systemType| \texttt{'}|onePolynomial|})
 (\texttt{|htpSetProperty| htPage \texttt{'}|exitFunction| \texttt{'}|bcInputSolveInfo|})
 (\texttt{|bcInputEquations| htPage \texttt{'}|exact|}))
\end{verbatim}

---
defun bcSystemSolveEqns1

[htpSetProperty p1408]
[bcInputEquations p1385]

— defun bcSystemSolveEqns1 —

(defun |bcSystemSolveEqns1| (htPage)
  (htpSetProperty htPage '|systemType| '|polynomial|)
  (htpSetProperty htPage '|exitFunction| '|bcInputSolveInfo|)
  (bcInputEquations htPage '|exact|))

—

defun bcLinearSolveEqns1

[htpSetProperty p1408]
[bcInputEquations p1385]

— defun bcLinearSolveEqns1 —

(defun |bcLinearSolveEqns1| (htPage)
  (htpSetProperty htPage '|systemType| '|linear|)
  (htpSetProperty htPage '|exitFunction| '|bcLinearSolveEqnsGen|)
  (bcInputEquations htPage '|exact|))

—

defun bcInputSolveInfo

[htInitPage p1415]
[httpPropertyList p1408]
[htpSetProperty p1408]
[htInputAreaList p??]
[htMakePage p1416]
[htShowPage p1416]

— defun bcInputSolveInfo —

(defun |bcInputSolveInfo| (htPage)
  (let (page)
    (setq page (htInitPage "Solve Basic Command" (httpPropertyList htPage)))
    (htSetProperty page '|numberOfEquations|)
    (htProperty htPage '|numberOfEquations|)))
42.2. FUNCTIONS CREATING PAGES

```lisp
(defun bcInputEquations
  (strconc p)
  (bcMakeLinearEquations p)
  (bcMakeEquations p)
  (htProperty p)
  (parse-integer p)
  (objValUnwrap p)
  (htInitPage p)
  (htpPropertyList p)
  (htpSetProperty p)
  (htSay p)
  (htMakePage p)
  (bcHt p)
  (bcMakeUnknowns p)
  (htMakeDoneButton p))
```

---

```
()!="htpSetProperty| page 'inputArea| (htpInputAreaAlist| htPage))
(=htMakePage
  !(domainConditions| (isDomain| PI (PositiveInteger|)))
  (text . "What would you like?"
  (text . "\beginmenu" (text . "\item "))
  (bcLinks
    '"menuitemstyle{Exact Solutions}" "bcSolveEquations| exact|)
  (text . "\indentrel{18}\tab{0} ")
  (text . "Solutions expressed in terms of {\em roots} of irreducible polynomials")
  (text . "\indentrel{-18}")
  (bcLinks
    '"menuitemstyle{Numeric Solutions}" "bcSolveEquationsNumerically| numeric|)
  (text . "\indentrel{18}\tab{0} ")
  (text . "Solutions expressed in terms of approximate real or complex {\em numbers}"
  (text . "\indentrel{-18}")
  (text . "\indentrel{-18}")
  (text . "\item "))
  (bcLinks
    '"menuitemstyle{Radical Solutions}" "bcSolveEquations| radical|)
  (text . "\indentrel{18}\tab{0} ")
  (text . "Solutions expressed in terms of {\em radicals} if it is possible"
  (text . "\indentrel{-18}")
  (text . "\endmenu")))
(=htShowPage)))
```
<table>
<thead>
<tr>
<th>defun bcInputEquations</th>
</tr>
</thead>
</table>

(defun |bcInputEquations| (htPage solutionMethod)
(labels (f (i n linearp)
  (let (spacer prefix lnam rnam var)
    (setq spacer (cond ((> i 99) 0) ((> i 9) 1) (t 2))
    (setq prefix
      (strconc "\newline\tab{2}{\em Equation " (princ-to-string i) ":"})))
    (setq lnam (intern (strconc "l" (princ-to-string i))))
    (setq rnam (intern (strconc "r" (princ-to-string i))))
    (setq var (if linearp
      (|bcMakeLinearEquations| i n)
      (|bcMakeEquations| i n)))
    (cons
      (cons '|text| prefix)
      (list (list '|bcStrings| (list 30 var lnam 'p))
        '|text| . " = ")
      (list '|bcStrings| (list 5 '|0| rnam 'p)))))))
(let (numEqs linearPred labelList equationPart page)
  (declare (special |$EmptyMode| |$bcParseOnly|))
  (setq numEqs
    (cond
      ((eq (|htpProperty| htPage '|systemType|) '|onePolynomial|) 1)
      (|$bcParseOnly|
        (parse-integer (|htpLabelInputString| htPage '|numberOfEquations|)))
      (t
        (|objValUnwrap| (|htpLabelSpadValue| htPage '|numberOfEquations|))))
  (setq linearPred (eq (|htpProperty| htPage '|systemType|) '|linear|))
  (setq labelList
    (cond
      ((eql numEqs 1)
        '((|bcStrings| (42 "x^2+1" 11 p)) '|text| . " = ")
        (|bcStrings| (6 0 r1 P))))
      (t
        (loop for i from 1 to numEqs
          append (f i numEqs linearPred)))))))
  (setq equationPart
    (cons
      (|domainConditions|
        (|isDomain| P (|Polynomial| |$EmptyMode|))
        (|isDomain| S (|String|))
        (|isDomain| PI (|PositiveInteger|))
        labelList))
  (setq page (|htInitPage| "Solve Basic Command" (|htPropertyList| htPage))))
42.2. FUNCTIONS CREATING PAGES

((|htSetProperty| page 'numberOfEquations| numEqs)
((|htSetProperty| page 'solutionMethod| solutionMethod)
((|htSay| "newline\menuitemstyle{}\tab{2}"
((|htSay| (if (eql numEqs 1)
   "Enter the {\em Equation}:
   "Enter the {\em Equations}:
))
((|htMakePage| equationPart)
((|bcHt| "\blankline ")
((|htSay| "newline\menuitemstyle{}\tab{2}"
((|htMakePage|
  (if (eql numEqs 1)
    '((|text| . "Enter the {\em unknown} (leave blank if implied): ")
      (|text| . "\tab{48}
       (|bcStrings| (6 "x" unknowns S . |quoteString|))))
    (list
     '((|text| . "Enter the unknowns (leave blank if implied):")
      '(|text| . "\tab{44}"
       (list '|bcStrings|
         (list 10 (|bcMakeUnknowns| numEqs 'unknowns| 'p))))))
  (|htMakeDoneButton| "Continue" '|bcInputEquationsEnd|)
  (|htShowPage|))))))

---

defun Create a variable string

--- defun bcCreateVariableString ---

(defun |bcCreateVariableString| (i)
  (format nil "x~a" i))

---

defun bcMakeUnknowns

--- defun bcMakeUnknowns ---

(defun |bcMakeUnknowns| (number)
  (format nil ""{"A"}"
    (loop for i from 1 to number collect (format nil "x~a " i))))

---
defun bcMakeEquations

(strconc p??)
[bcCreateVariableString p1387]
[nreverse0 p??]

— defun bcMakeEquations —

(defun |bcMakeEquations| (i number)
(if (eql number 1)
  (strconc ([bcCreateVariableString| 1] ' ¦^2+1))
(progn
  ([bcCreateVariableString| i])
  (strconc
   (strconc (apply 'concat
     (let (t1)
       (do ((j 1 (1+ j))) ((> j number) (nreverse0 t1))
         (setq t1 (cons (strconc ([bcCreateVariableString| j] ' (+) t1)))
                   '1))
       (strconc '2* (strconc ([bcCreateVariableString| i] ' ¦^2)))))))

——

defun bcMakeLinearEquations

[bcCreateVariableString p1387]
[strconc p??]
[nreverse0 p??]

— defun bcMakeLinearEquations —

(defun |bcMakeLinearEquations| (i number)
(cond
  ((eql number 1) ([bcCreateVariableString| 1])
  ((eql number 2)
   (cond
    ((eql i 1)
     (strconc ([bcCreateVariableString| 1]
       (strconc ' + ([bcCreateVariableString| 2])))
    (t
     (strconc ([bcCreateVariableString| 1]
       (strconc ' - ([bcCreateVariableString| 2])))
    (t
     (strconc
      (strconc
       (apply 'concat
        (let (t1)
(do ((j 1 (1+ j))) ((> j number) (nreverse0 t1))
    (setq t1 (cons (strconc (|bcCreateVariableString| j) '+) t1)))
'(|i|)
(strconc '(-2* (|bcCreateVariableString| i)))))))

---

defun bcInputEquationsEnd

If exitFunction is set, call it. [systemError p??]

---

(defun |bcInputEquationsEnd| (htPage)
  (let (fun)
    (if (setq fun (|htpProperty| htPage '|exitFunction|))
      (funcall fun htPage)
      (|systemError| nil))))

---

defun bcSolveEquationsNumerically

[htInitPage p1415]
[htMakePage p1416]
[htMakeDoneButton p1437]
[htShowPage p1416]
[htpPropertyList p1408]

---

(defun |bcSolveEquationsNumerically| (htPage p)
  (declare (ignore p))
  (|htInitPage| "Solve Basic Command" (|htpPropertyList| htPage))
  (|htMakePage|'
    '((|text| . "What would you like?")
      (|radioButtons| |choice|
        ("Real roots expressed as rational numbers" "|rr|
        )
        ("Real roots expressed as floats" "|rf|
        )
        ("Complex roots expressed as rational numbers" "|cr|
        )
        ("Complex roots expressed as floats" "|cf|
        )
      )
      (|text| . "\vspace{1}\newline")
      (|inputStrings| ("Enter the number of desired \em digits of accuracy" "5 20 |acc| PI))))
  (|htMakeDoneButton| "Continue" '|bcSolveNumerically1|))
defun bcSolveNumerically1

(defun bcSolveNumerically1 (htPage)
  (bcSolveEquations htPage 'numeric))

---

defun bcSolveEquations

(defun bcSolveEquations (htPage solutionMethod)
  (let (digits kind accString alist varpart r varlist varString eqnString name)
    (when (eq solutionMethod 'numeric)
      (setq digits (htpLabelInputString htPage '|acc|))
      (setq kind (htpButtonValue htPage '|choice|))
      (setq accString
        (if (member kind '(rf |cf|))
          (strconc "1.e-" digits)
          (strconc "1/10**" digits))))
      (setq alist (htpProperty htPage '|inputArea|))
      (setq varpart (cadar alist))
      (setq r (cdr alist))
      (setq varlist (bcString2WordList varpart))
      (setq varString
        (if (cdr varlist)
"
defun Linear Solve Basic Command trampoline

[bcReadMatrix p1336]
This routine is a trampoline. It calls bcReadMatrix passing the name of a call-back routine bcLinearSolveMatrix1 to be called after the matrix has been read.

— defun bcLinearSolveMatrix —

(defun bcLinearSolveMatrix (htPage junk)
  (declare (ignore htPage junk))
  (|bcReadMatrix| "|bcLinearSolveMatrix1|"))

—

defun Linear Solve Basic Command options

[htInitPage p1415]
[htMakePage p1416]
[htShowPage p1416]
[$EmptyMode p640]

The right side vector B is:

  Zero: the system is homogeneous
  Not zero: the system is not homogeneous

— defun bcLinearSolveMatrix1 —

(defun bcLinearSolveMatrix1 (htPage)
  (let (page)
    (setq page
      (|htInitPage| "Linear Solve Basic Command" (|htpPropertyList| htPage)))
      (|htpSetProperty| page '|matrix| (|bcLinearExtractMatrix| htPage)))

—
CHAPTER 42. HYPERDOC BASIC COMMAND SUPPORT

defun bcLinearExtractMatrix
[htpInputAreaAlist p1407]

— defun bcLinearExtractMatrix —

(defun |bcLinearExtractMatrix| (htPage)
  (reverse (|htpInputAreaAlist| htPage)))

defun Linear Solve Basic Command options
[strconc p??]
[htpProperty p1408]
[htInitPage p1415]
[htpPropertyList p1408]
[htpSetProperty p1408]
[htMakePage p1416]
[htShowPage p1416]
[$EmptyMode p640]

Enter the right side vector B:
  Coefficient 1: 0
  Coefficient 2: 0

Do you want:
  All the solutions?
  A particular solution?

— defun bcLinearSolveMatrixInhomo —

(defun |bcLinearSolveMatrixInhomo| (htPage junk)
(declare (ignore junk))
(labels (f i)
  (let (spacer prefix name)
    (setq spacer (cond ((> i 99) 0) ((> i 9) 1) (t 2)))
    (setq prefix (strconc "{\em Coefficient " (princ-to-string i) ":}"))
    (unless (eql spacer 0)
      (setq prefix (strconc prefix \space{" (princ-to-string spacer) "})))
    (setq name (intern (strconc "c" (princ-to-string i))))
    (list prefix '|| 30 0 name 'p ))))
(let (nrows ncols labelList page)
  (declare (special |$EmptyMode|))
  (setq nrows (htpProperty htPage '|nrows|))
  (setq ncols (htpProperty htPage '|ncols|))
  (setq labelList (loop for i from 1 to ncols collect (f i)))
  (setq page
    (htInitPage "Linear Solve Basic Command" (htpPropertyList htPage)))
  (htpSetProperty page '|matrix| (htpProperty htPage '|matrix|))
  (htpSetProperty page '|nrows| nrows)
  (htpSetProperty page '|ncols| ncols)
  (htMakePage)
    (list
      '|(domainConditions| (|isDomain| P (|Polynomial| |$EmptyMode|)))
      '|(text| . "Enter the right side vector B:")
      (cons 'inputStrings labelList)
      (list
        '|(text| . \vspace{1}\newline Do you want:")
        (cons nil 'list)
        (cons (lispLinks
          ("All the solutions?" " |bcLinearSolveMatrixInhomoGen| |all|)
          ("A particular solution?" " |bcLinearSolveMatrixInhomoGen| |particular|)))
        )))
  (htShowPage))

defun bcLinearSolveMatrixInhomoGen

[bcLinearMatrixGen p1394]

--- defun bcLinearSolveMatrixInhomoGen ---

(defun |bcLinearSolveMatrixInhomoGen| (htPage key)
  (|bcLinearMatrixGen| htPage key))

---
defun bcLinearSolveMatrixHomo

(defun bcLinearSolveMatrixHomo (htPage key)
  (declare (ignore key))
  (|bcLinearMatrixGen| htPage '|homo|)
)

defun bcLinearMatrixGen

(defun bcLinearMatrixGen (htPage key)
  (let (matform vector vecform form)
    (setq matform (|bcMatrixGen| htPage))
    (cond
      ((eq key '|homo|)
       (|bcFinish| "nullSpace" matform))
      (t
       (setq vector
         (loop for x in (reverse (|htpInputAreaAlist| htpage))
               collect (elt x 1)))
       (setq vecform (|bcVectorGen| vector))
       (setq form ((|bcMkFunction| "solve" matform (cons vecform nil)))
         (|bcGen| (if (eq key '|particular|)
                   (strconc form ".particular")
                   form)))))))
defun linearFinalRequest

[sayBrightly "???
[bcQueryInteger "???
[explainLinear

— defun linearFinalRequest —

(defun linearFinalRequest (nhh mat vect)
  (declare (ignore mat vect))
  (let (tt)
    (sayBrightly "Do you want more information on the meaning of the output")
    (sayBrightly " (1) no ")
    (sayBrightly " (2) yes ")
    (setq tt (bcQueryInteger 1 2 t))
    (cond
      ((eql tt 1) (sayBrightly "Bye Bye"))
      ((eql tt 2) (explainLinear nhh))))

defun explainLinear

[systemError

— defun explainLinear —

(defun explainLinear (flag)
  (cond
    ((eq flag 'notHomogeneous)
      ("solve returns a particular solution and a basis for"
       "the vector space of solutions for the homogeneous part."
       "The particular solution is \"failed\" if one cannot be found.\")
    ((eq flag 'homogeneous)
      ("solve returns a basis for"
       "the vector space of solutions for the homogeneous part")
      (t (systemError nil))))

defun finalExactRequest

[bcQueryInteger "???
[sayBrightly "???
[moreExactSolution "???
--- defun finalExactRequest ---

(defun finalExactRequest (equations unknowns)
  (let (tt)
    (|sayBrightly| "Do you like:
    (1) the solutions how they are displayed"
    (2) to get ???")
    (setq tt (|bcQueryInteger| 1 3 t))
    (cond
      ((eql tt 1) (|sayBrightly| "Bye Bye"))
      ((eql tt 2) (|moreExactSolution| equations unknowns))
      ((eql tt 3) (|explainExact| equations unknowns))))

---

defun bcLinearSolveEqnsGen

(defun bcLinearSolveEqnsGen (htPage)
  (let (vars varlist varString alist eqnString)
    (setq alist (|htpInputAreaAlist| htPage))
    (when (setq vars (|htpLabelInputString| htPage '|unknowns|))
      (setq varlist (|bcString2WordList| vars))
      (setq varString
        (if (cdr varlist) (|bcwords2liststring| varlist) (car varlist)))
      (setq alist (cdr alist)))
    (setq eqnString (|bcGenEquations| alist))
    (|bcFinish| "solve" eqnString varString))

---

defun bcGenEquations

(defun bcGenEquations
  (strconc ??)
  (|bcwords2liststring| p1400)
42.2. FUNCTIONS CREATING PAGES

— defun bcGenEquations —

(defun bcGenEquations (alist)
  (let (right left y eqnlist)
    (setq y alist)
    (loop while y do
      (setq right (elt (car y) 1))
      (setq y (cdr y))
      (setq left (elt (car y) 1))
      (setq y (cdr y))
      (setq eqnlist (cons (strconc left " = " right) eqnlist)))))

(defun bcFinish (&rest t1 &aux args arg name)
  (dsetq (name arg . args) t1)
  (bcGen (bcMkFunction name arg args)))

— defun bcMkFunction —

(defun bcMkFunction (name arg args)
  (let (str)
    (setq str
      (let ((result "")
        (dolist (i args result)
          (when i
            (setq result (concatenate 'string result (concatenate 'string "," i)))))
        (concatenate 'string name "(" str ")")))
      (concatenate 'string arg))
    "test(arg1,arg2,arg3)"

defun Output the final formula

defun convert arguments into function call syntax

Convert verb—(bcMkFunction "test" "arg1" "arg2" "arg3")—to "test(arg1,arg2,arg3)"
defun bcString2HyString2

— defun bcString2HyString2 —

(defun bcString2HyString2 (s)
  (if (and (stringp s) (char= (elt s 0) #\"))
      (concatenate 'string "\\" s "\\")
    s))

———

defun bcString2HyString

— defun bcString2HyString —

(defun bcString2HyString (s) s)

———

defun find a character position in a string

— defun bcFindString —

(defun bcFindString (s i n char)
  (position char s :start i :end n))

———

defun Basic Command result page

[conclude p??]
[htInitPage p1415]
[htMakePage p1416]
[htShowPage p1416]

— defun bcGen —

(defun bcGen (command)
  (let (string)
42.2. FUNCTIONS CREATING PAGES

([htInitPage] "Basic Command" nil)
(setq string
  (if (< (length command) 50)
    (strconc "\centerline{\tt " command " }")
    (strconc "\tt " command " "))))
([htMakePage]
  (list
    '([text] "{Here is the AXIOM command you could have issued to compute this result:}"
      "\vspace{2}\newline ")
    (cons '([text] string) )))
([htMakeDoitButton] "Do It" command)
([htShowPage]))

-----

defun Basic Command result page – NAG version

[strconc p??]
[htInitPage p1415]
[htMakePage p1416]
[htShowPage p1416]
Except for the banner the bcGen and linkGen functions are identical. We no longer care so
we just call bcGen.

— defun linkGen —

(defun |linkGen| (command)
  (|bcGen| command))

-----

defun bcOptional

— defun bcOptional —

(defun |bcOptional| (s)
  (if (string-equal s "") "2" s))

-----
defun create a vertical space on a page

— defun bcvspace —

(defun |bcvspace| ()
  (|bcHt| "\vspace{1}\newline "))

—

defun break a string into words

— defun bcString2WordList —

(defun |bcString2WordList| (string)
  (loop for i = 0 then (1+ j)
    as j = (position #\space string :start i)
    collect (subseq string i j)
    while j))

—

defun format words into a string

[strconc p1400]

— defun bcwords2liststring —

(defun |bcwords2liststring| (words)
  (format nil "["~{~A~^, ~}" words))

—

defun format a vector

[bcwords2liststring p1400]

— defun bcVectorGen —

(defun |bcVectorGen| (vec)
  (|bcwords2liststring| vec))
defun format an error message

[sayBrightlyNT p??]
[sayBrightly p??]

— defun bcError —

(defun bcError (string)
  (|sayBrightlyNT| "NOTE: ")
  (|sayBrightly| string))

defun format intervals

[strconc p??]

— defun bcDrawIt —

(defun bcDrawIt (ind a b)
  (strconc ind "=" a ".." b))

defun Basic Command page not ready

[htInitPage p1415]
[htMakePage p1416]
[htShowPage p1416]

— defun bcNotReady —

(defun bcNotReady (htPage)
  (declare (ignore htPage))
  (|htInitPage| "Basic Command" NIL)
  (|htMakePage|
    '(((|text| . "{\centerline{\em This facility will soon be available}}"))))
  (|htShowPage|))
**defun pad a string with blanks**

```lisp
(defun pad-string (str n w)
  (let ((s ws))
    (setq s (princ-to-string n))
    (setq ws (len s))
    (strconc "\space{" (princ-to-string (1+ (- w ws))) "}" s)))
```

---

**defun construct a name string**

Given ("one" "two" "three") generate ")one,two,three)"

```lisp
(defun stringList2String (x)
  (let (str)
    (cond
      ((null x) "()")
      (t
    (setq str
      (let ((result ""))
        (concatenate 'string (car x)
          (dolist (i (cdr x) result)
            (setq result (concatenate 'string result
              (concatenate 'string "," i))))))))
      (concatenate 'string "(" str ")))))))
```

---

**defun construct a name string**

```lisp
(defun htMkName (s n)
  (strconc s (princ-to-string n)))
```

---

`;; ht-util merge`
defvar $bcParseOnly

    — initvars —

    (defvar |$bcParseOnly| t)

———

defvar $htLineList

    — initvars —

    (defvar |$htLineList| nil)

———

defvar $curPage

    — initvars —

    (defvar |$curPage| nil)

———

defvar $activePageList

    — initvars —

    (defvar |$activePageList| nil)

———

defun htpDestroyPage

    — defun htpDestroyPage —
(defun htpDestroyPage (pageName)
  (declare (special $activePageList))
  (SEQ (cond
    ((member pageName $activePageList)
      (EXIT (progn
        (set pageName nil)
        (setq $activePageList
          (delete $activePageList pageName :test #'equal))))))))

HTPAGE STRUCTURE

This is a list with the fields

1. name
2. Domain Conditions
3. Domain Variable Alist
4. Domain Pvar Subst List
5. Radio Button Alist
6. Input Area Alist
7. Property List
8. Description

(defun htpName

  — defun htpName —

(defun htpName (htPage) (elt htPage 0))

——

(defun htpSetName

  — defun htpSetName —

(defun htpSetName | defun htpSetName |
42.2. FUNCTIONS CREATING PAGES

(defun htpSetName (htPage val) (setelt htPage 0 val))

(defun htpDomainConditions
  (defun htpDomainConditions —
  (defun htpDomainConditions| (htPage) (elt htPage 1)))

(defun htpSetDomainConditions
  (defun htpSetDomainConditions —
  (defun htpSetDomainConditions| (htPage val)
    (setelt htPage 1 val)))

(defun htpDomainVariableAlist
  (defun htpDomainVariableAlist —
  (defun htpDomainVariableAlist| (htPage) (elt htPage 2)))

(defun htpSetDomainVariableAlist
  (defun htpSetDomainVariableAlist —
  (defun htpSetDomainVariableAlist| (htPage val)
    (setelt htPage 2 val)))
defun htpDomainPvarSubstList

    — defun htpDomainPvarSubstList —

(defun htpDomainPvarSubstList (htPage) (elt htPage 3))

———

defun htpSetDomainPvarSubstList

    — defun htpSetDomainPvarSubstList —

(defun htpSetDomainPvarSubstList (htPage val)
  (setelt htPage 3 val))

———

defun htpRadioButtonAlist

    — defun htpRadioButtonAlist —

(defun htpRadioButtonAlist (htPage) (elt htPage 4))

———

defun htpButtonValue

    — defun htpButtonValue —

(defun htpButtonValue (htPage groupName)
  (prog ()
    (return
      (SEQ (DO ((G166092 (LASSOC groupName (htpRadioButtonAlist htPage))
        (CDR G166092))
        (buttonName nil))
      ((OR (ATOM G166092)
        (OR (ATOM (G166092)
         (progn (setq buttonName (car G166092)) nil))))))
NIL)
(SEQ (EXIT (COND
  ((BOOT-EQUAL
     (|stripSpaces|
     (|htpLabelInputString| htPage
     |buttonName|))
     "t")
     (EXIT (RETURN |buttonName|))))))))))

defun htpSetRadioButtonAlist

— defun htpSetRadioButtonAlist —

(defun |htpSetRadioButtonAlist| (htPage val)
  (setelt htPage 4 val))

———

defun htpInputAreaAlist

— defun htpInputAreaAlist —

(defun |htpInputAreaAlist| (htPage)
  (elt htPage 5))

———

defun htpSetInputAreaAlist

— defun htpSetInputAreaAlist —

(defun |htpSetInputAreaAlist| (htPage val)
  (setelt htPage 5 val))

———
defun htpAddInputAreaProp

— defun htpAddInputAreaProp —

(defun htpAddInputAreaProp (htPage label prop)
  (setelt htPage 5
    (cons
      (cons label (cons nil (cons nil (cons nil prop))))
      (elt htPage 5)))))

——

defun htpPropertyList

— defun htpPropertyList —

(defun htpPropertyList (htPage)
  (elt htPage 6))

——

defun htpProperty

— defun htpProperty —

(defun htpProperty (htPage propName)
  (assoc propName (elt htPage 6)))

——

defun htpSetProperty

— defun htpSetProperty —

(defun htpSetProperty (htPage propName val)
  (let (pair)
    (setq pair (assoc propName (elt htPage 6)))
    (cond
      (pair (rplacd pair val))
      (t (setelt htPage 6 (cons (cons propName val) (elt htPage 6)))))
defun htpLabelInputString

— defun htpLabelInputString —

(defun htpLabelInputString (htPage label)
  (let (props s)
    (setq props (assoc label (htpInputAreaAlist htPage)))
    (when (and props (stringp (setq s (elt props 0))))
      (if (equal s "") s (trimString s))))

defun htpLabelFilteredInputString

— defun htpLabelFilteredInputString —

(defun htpLabelFilteredInputString (htPage label)
  (let (props)
    (setq props (assoc label (htpInputAreaAlist htPage)))
    (when props
      (cond
        ((and (> (length props) 5) (elt props 6))
          (funcall (symbol-function (elt props 6)) (elt props 0)))
        (t (replacePercentByDollar (elt props 0))))))

defun replacePercentByDollar,fn

— defun replacePercentByDollar,fn —

(defun replacePercentByDollar,fn (s i n)
  (let (m)
    (cond
      ( (> i n) "")
      ((> (setq m (char-position #\% s i)) n) (substring s i nil))
      (t (strconc (substring s i (- m i)) "$”)
        (replacePercentByDollar,fn s (1+ m n))))))
defun replacePercentByDollar

--- defun replacePercentByDollar ---
(defun replacePercentByDollar (s)
  (replacePercentByDollar fn s 0 (maxindex s)))

---
defun htpSetLabelInputString

--- defun htpSetLabelInputString ---
(defun htpSetLabelInputString (htPage label val)
  (let (props)
    (setq props (lassoc label (htpInputAreaAlist htPage)))
    (when props (setelt props 0 (princ-to-string val)))))

---
defun htpLabelSpadValue

--- defun htpLabelSpadValue ---
(defun htpLabelSpadValue (htPage label)
  (let (props)
    (setq props (lassoc label (htpInputAreaAlist htPage)))
    (when props (elt props 1)))))

---
defun htpSetLabelSpadValue

--- defun htpSetLabelSpadValue ---
(defun htpSetLabelSpadValue (htPage label val)
  (let (props)
    (setq props (lassoc label (htpInputAreaAlist htPage)))
    (when props (setelt props 1 val))))
42.2. FUNCTIONS CREATING PAGES

---

defun htpLabelErrorMsg

— defun htpLabelErrorMsg —

(defun htpLabelErrorMsg (htPage label)
  (let (props)
    (setq props (lassoc label (htpInputAreaAlist htPage)))
    (when props (elt props 2)))

---

defun htpSetLabelErrorMsg

— defun htpSetLabelErrorMsg —

(defun htpSetLabelErrorMsg (htPage label val)
  (let (props)
    (setq props (lassoc label (htpInputAreaAlist htPage)))
    (when props (setelt props 2 val)))

---

defun htpLabelType

— defun htpLabelType —

(defun htpLabelType (htPage label)
  (let (props)
    (setq props (lassoc label (htpInputAreaAlist htPage)))
    (when props (elt props 3)))

---

defun htpLabelDefault

— defun htpLabelDefault —
(defun htpLabelDefault (htPage label)
  (let (msg props)
    (cond
      ((setq msg (htpLabelInputString htPage label))
        (cond
          ((equal msg "t") 1)
          ((equal msg "nil") 0)
          (t msg))
      (t
       (setq props (lassoc label (htpInputAreaAlist htPage)))
       (when props (elt props 4)))))

---

defun htpLabelSpadType

---

defun htpLabelSpadType (htPage label)
  (let (props)
    (setq props (lassoc label (htpInputAreaAlist htPage)))
    (when props (elt props 5)))

---

defun htpLabelFilter

---

defun htpLabelFilter (htPage label)
  (let (props)
    (setq props (lassoc label (htpInputAreaAlist htPage)))
    (when props (elt props 6)))

---

defun htpPageDescription

---

defun htpPageDescription
defun htpPageDescription

— defun htpPageDescription —

(defun htpPageDescription (htPage)
  (elt htPage 7))

———

defun htpSetPageDescription

— defun htpSetPageDescription —

(defun htpSetPageDescription (htPage pageDescription)
  (setelt htPage 7 pageDescription))

———

defun htpAddToPageDescription

— defun htpAddToPageDescription —

(defun htpAddToPageDescription (htPage pageDescrip)
  (setelt htPage 7
    (nconc (nreverse (copy-list pageDescrip)) (elt htPage 7))))

———

defun issue a single hypertex line or group of lines

— defun iht —

(defun iht (line)
  (declare (special $htLineList $newPage))
  (cond
    ($newPage nil)
    ((consp line)
      (setq $htLineList
        (nconc (nreverse (mapStringize (copy-list line)))
        $htLineList)))
    (t
      (setq $htLineList
        (cons (basicStringize line) $htLineList))))

———
defun bcHt

— defun bcHt —

(defun bcHt (line)
  (declare (special $curPage $newPage))
  (progn
    (iht line)
    (cond
      ((consp line)
       (cond
         (($newPage)
          (htpAddToPageDescription $curPage
           (cons (cons 'text line) nil)))
         (t nil)))
       ($newPage)
       (htpAddToPageDescription $curPage
        (cons (cons 'text (cons line nil)) nil)))
       (t nil))))

defun bcIssueHt

— defun bcIssueHt —

(defun bcIssueHt (line)
  (cond ((consp line) (htMakePage1 line)) (t (iht line))))

defun mapStringize

— defun mapStringize —

(defun mapStringize (z)
  (cond
    ((atom z) z)
    (t (rplaca z (basicStringize (car z)))
      (rplacd z (mapStringize (cdr z))))))
defun basicStringize

    — defun basicStringize —

(defun basicStringize (s)
  (cond ((stringp s)
       (cond ((equal s "$\$") "\%")
             ((equal s "{$em\$}") "{$em\%}\"")
             (t s)))
       ((eq s '$) "\%")
       (t (princ-to-string s)))))

defun stringize

    — defun stringize —

(defun stringize (s)
  (cond ((stringp s) s)
        (t (princ-to-string s)))))

defun htInitPage

    — defun htInitPage —

(defun htInitPage (title propList)
  (declare (special $curPage))
  (progn
    (htInitPageNoScroll propList title)
    (htSayStandard "$beginscroll ")
    ($curPage)))

defun htAddHeading

    — defun htAddHeading —
(defun \htAddHeading\ (title)
 (declare (special \$curPage\))
 (\htNewPage\ title)
 \$curPage\)

——

defun \htShowPage\ —

defun \htShowPage\ ()
 (\htSayStandard\ "\endscroll")
 (\htShowPageNoScroll\))

——

defun show the page which has been computed

defun \htShowPageNoScroll\ —

defun \htShowPageNoScroll\ ()
 (let (line)
 (declare (special \$htLineList\ \$curPage\ \$newPage\))
 (\htSayStandard\ "\autobuttons")
 (\htpSetPageDescription\ \$curPage\)
 (\nreverse\ (\htpPageDescription\ \$curPage\)))
 (setq \$newPage\ nil)
 (setq \$htLineList\ nil)
 (\htMakePage\ (\htpPageDescription\ \$curPage\))
 (setq line (apply #'concat (\nreverse\ \$htLineList\)))
 (\issueHT\ line)
 (\endHTPage\)))

——

defun make a page given the description in itemList

defun \htMakePage\ —
defun htMakePage1

(defun htMakePage1 (itemList)
 (prog (itemType items)
  (return
   (SEQ (DO ((G166261 itemList (CDR G166261)) (G166253 NIL))
     ((OR (ATOM G166261)
       (PROGN (SETQ G166253 (CAR G166261)) NIL)
       (PROGN
         (PROGN
           (setq itemType (CAR G166253))
           (setq items (CDR G166253))
           G166253)
       NIL))
     NIL)
   (SEQ (EXIT (COND
     ((eq itemType '|text|) (iht items))
     ((eq itemType '|lispLinks|) (htLispLinks items))
     ((eq itemType '|lispmemoLinks|) (htLispMemoLinks items))
     ((eq itemType '|bcLinks|) (htBcLinks items))
     ((eq itemType '|bcLinksNS|) (htBcLinks items t))
     ((eq itemType '|bcLispLinks|) (htBcLispLinks items))
     ((eq itemType '|radioButtons|) (htRadioButtons items))
     ((eq itemType '|bcRadioButtons|) (htBcRadioButtons items))
     ((eq itemType '|inputStrings|) (htInputStrings items))
     ((eq itemType '|domainConditions|) (htProcessDomainConditions items))))
   (progn (cond
     (|$newPage| ([htpAddToPageDescription| |$curPage| itemList)))
     (|htMakePage1| itemList)))))

42.2. FUNCTIONS CREATING PAGES
defun htMakeErrorPage

— defun htMakeErrorPage —

(defun htMakeErrorPage (htPage)
  (prog (line)
    (declare (special $curPage $htLineList $newPage))
    (return
      (progn
        (setq $newPage nil)
        (setq $htLineList nil)
        (setq $curPage htPage)
        (|htMakePage| (|htpPageDescription| htPage))
        (setq line (apply #'CONCAT (NREVERSE $htLineList)))
        (|issueHT| line)
        (|endHTPage|))))

defun htQuote

— defun htQuote —

(defun htQuote (s)
  (|iht| "\\")
  (|iht| s)
defun htProcessToggleButtons

— defun htProcessToggleButtons —

(defvar htPICKT "\item{\em\inputbox[\htpLabelDefault $\curPage$ buttonName]{\htbmfile{pick}}{\htbmfile{unpick}}\space{}}"

(htpIssueHt message)
(htiht "\space{}}")
(htpIssueHt info))))

(defun htProcessToggleButtons (buttons)
  (prog (message info defaultValue buttonName)
    (declare (special $curPage$))
    (return
      (seq (prog
        (hlht "\newline\indent{5}\beginitems ")
        (do ((G166302 buttons (cdr G166302))
             (G166286 nil))
             ((or (atom G166302)
                 (setq G166286 (car G166302)) nil)
               nil)
        (seq (exit (prog
          (cond
            ((null (assoc buttonName
                          (htInputAreaAlist $curPage$)))
             (setUpDefault buttonName
               (cons $\curPage$ buttonName
                 (cons defaultValue nil)))))
            (hlht "\item{\em\inputbox["
              (cons
                (hlhtLabelDefault $\curPage$ buttonName)
                (cons "}"
                  (cons buttonName
                    (cons "}{\htbmfile{pick}}{{\htbmfile{unpick}}}{\space{}}"
                      nil))))))))
          (htpIssueHt message)
          (hlht "\space{}}")
          (htpIssueHt info))))))

defun htProcessBcButtons

— defun htProcessBcButtons —

(defun htProcessBcButtons (buttons)
  (prog (defaultValue buttonName k)
    (declare (special|$curPage|))
    (return
      (SEQ (DO ((G166328 buttons (CDR G166328)) (G166317 nil))
          ((OR (ATOM G166328)
            (progn (setq G166317 (car G166328)) nil)
            (progn
              (setq defaultValue (car G166317))
              (setq buttonName (CADR G166317))
              G166317)
            nil))
      nil)
      (SEQ (EXIT (progn
          (cond
            ((NULL (LASSOC buttonName
              (htpInputAreaAlist|$curPage|)))
             (setUpDefault buttonName
               (cons '|button|
                 (cons defaultValue nil))))))
            (setq k
              (htpLabelDefault|$curPage|
               buttonName))
            (cond
              ((EQL k 0)
               (lht| (cons "\\off{"
                 (cons buttonName
                   (cons "}" nil))))))
              ((EQL k 1)
               (lht| (cons "\\on{"
                 (cons buttonName
                   (cons "}" nil))))))
            (t
               (lht| (cons "\\inputbox["
                 (cons
                   (htpLabelDefault|$curPage|
                    buttonName)
                   (cons "}" nil))))))
      nil)))
  )
  )
defun htProcessBcStrings

— defun htProcessBcStrings —

(defun htProcessBcStrings (strings)
  (prog (numChars default stringName spadType filter mess2)
    (declare (special curPage))
    (return
      (seq (do ((g2 strings (cdr g2)) (G166343 nil))
          ((or (atom g2)
              (progn (setq G166343 (car g2)) nil)
              (progn
                (progn
                  (setq numChars (car G166343))
                  (setq default (cadr G166343))
                  (setq stringName (caddr G166343))
                  (setq spadType (cadddr G166343))
                  (setq filter (cddddr G166343))
                  G166343)
                nil))
       nil)
    (seq (exit (progn
                  (setq mess2 "")
                  (cond
                    (null (assoc stringName (htInputAreaAlist curPage)))
                    (setUpDefault stringName
                      (cons 'string)
                      (cons default
                        (cons spadType
                          (cons filter nil)))))))
    (cond
      (null (assoc stringName
                  (htInputAreaAlist curPage)))
      (setUpDefault stringName
        (cons 'string)
        (cons default
          (cons spadType
            (cons filter nil))))))
  (seq (exit (progn
                  (setq mess2 "")
                  (cond
                    (null (assoc stringName
                        (htInputAreaAlist curPage)))
                    (setUpDefault stringName
                      (cons 'string)
                      (cons default
                        (cons spadType
                          (cons filter nil)))))))

defun htProcessBcStrings

— defun htProcessBcStrings —

(defun htProcessBcStrings (strings)
  (prog (numChars default stringName spadType filter mess2)
    (declare (special curPage))
    (return
      (seq (do ((g2 strings (cdr g2)) (G166343 nil))
          ((or (atom g2)
              (progn (setq G166343 (car g2)) nil)
              (progn
                (progn
                  (setq numChars (car G166343))
                  (setq default (cadr G166343))
                  (setq stringName (caddr G166343))
                  (setq spadType (cadddr G166343))
                  (setq filter (cddddr G166343))
                  G166343)
                nil))
       nil)
    (seq (exit (progn
                  (setq mess2 "")
                  (cond
                    (null (assoc stringName
                        (htInputAreaAlist curPage)))
                    (setUpDefault stringName
                      (cons 'string)
                      (cons default
                        (cons spadType
                          (cons filter nil))))))
    (cond
      (null (assoc stringName
                  (htInputAreaAlist curPage)))
      (setUpDefault stringName
        (cons 'string)
        (cons default
          (cons spadType
            (cons filter nil))))))
(setq mess2
  (concat mess2 (bcSadFaces)))
(htpSetLabelErrorMsg $curPage stringName nil))
(htpLabelDefault $curPage stringName)
(cons "\inputstring{" (cons numChars (cons "\}{")
(cons \beginitems)\enditems)\")))

---

defun bcSadFaces

— defun bcSadFaces —

(defun bcSadFaces ()
  "\space{1}{\em\htbitmap{error}\htbitmap{error}\htbitmap{error}}")

---

defun htLispLinks

— defun htLispLinks —

(defun htLispLinks (&REST G166422 &AUX option links)
  (setq links (car G166422))
  (setq option (cdr G166422))
  (prog (t1 options indent message info func value call)
    (return
     (SEQ (progn
       (setq t1 (|beforeAfter| '|'options| links))
       (setq links (car t1))
       (setq options (cadr t1))
       (setq indent (or (LASSOC '|indent| options) 5))
       (|ht| \newline\indent{")
       (|ht| \beginitems)\enditems")
       \beginitems)\enditems")
(DO ((G166403 links (CDR G166403)) (G166387 nil))
  ((or (atom G166403)
    (progn (setq G166387 (car G166403)) nil)
    (progn
      (setq message (car G166387))
      (setq info (cadr G166387))
      (setq func (caddr G166387))
      (setq value (cdddr G166387))
      G166387)
    nil))
  nil)
(SEQ (EXIT (progn
    (|iht| "$\item["
    (setq call
      (cond
        ((IFCAR option)
          "$\lispmemolink"
        (t "$\lispdownlink"))
        (|htMakeButton| call message
          (|mkCurryFun| func value))
        (|iht| (cons "$\\hspace{}" nil))
        (|bcIssueHt| info))))
    (|iht| "$\enditems\indent{0} "))))))

---

defun htLispMemoLinks

— defun htLispMemoLinks —

(defun |htLispMemoLinks| (links) (|htLispLinks| links t))

---

defun htBcLinks

— defun htBcLinks —

(defun |htBcLinks| (&rest a1)
  (let (skipStateInfo? t1 message info func value options links)
    (setq links (car a1)))
(setq options (cdr a1))
(setq skipStateInfo? (ifcar options))
(setq t1 (|beforeAfter| '|options| links))
(setq links (car t1))
(setq options (cadr t1))
(do ((g1 links (cdr g1)) (g2 nil))
((or (atom g1)
    (progn (setq g2 (car g1)) nil)
    (progn
      (setq message (car g2))
      (setq info (cadr g2))
      (setq func (caddr g2))
      (setq value (cdddr g2))
      g2)
    nil))
  nil)
(\lispdownlink message
(|mkCurryFun| func value) skipStateInfo?)
(|bcIssueHt| info)))

defun htBcLispLinks

— defun htBcLispLinks —

(defun |htBcLispLinks| (links)
  (prog (t1 options message info func value)
    (return
      (SEQ (progn
        (setq t1 (|beforeAfter| '|options| links))
        (setq links (car t1))
        (setq options (cadr t1))
        (DO ((G166487 links (cdr G166487)) (G166474 nil))
          ((or (atom G166487)
                (progn (setq G166487 (car G166487)) nil)
                (progn
                  (setq message (car G166474))
                  (setq info (cadr G166474))
                  (setq func (caddr G166474))
                  (setq value (cdddr G166474))
                  G166474)
                nil))
      nil))
    (SEQ (EXIT (progn
      nil)))))
### 42.2. FUNCTIONS CREATING PAGES

```lisp
(defun beforeAfter
  (x u)
  (prog (y r)
    (return
     (seq (cons (prog (G166514)
                   (setq G166514 nil)
                   (return
                    (dd ((G166504 u (cdr G166504)))
                    ((or (atom G166504)
                        (progn
                          (setq y (car G166504))
                          (setq r (cdr G166504))
                          G166504)
                        nil)
                    (null (nequal x y)))
                    (nreverse G166514))
                    (seq (exit (setq G166514 (cons y G166514)))))
                 (cons r nil))))))
```

### defun mkCurryFun

```lisp
(defun mkCurryFun
  (fun val)
  (prog (name code)
    (return
     (progn
       (setq name (gentemp))
       (setq code
         (cons 'defun
```
(defun htRadioButtons
  (G166546)
  (prog (groupName buttons boxesName message info buttonName defaultValue)
    (declare (special "$curPage"))
    (return
      (SEQ (progn
          (setq groupName (car G166546))
          (setq buttons (cdr G166546))
          (htpSetRadioButtonAlist "$curPage"
            (cons (cons groupName (buttonNames buttons))
              (htpRadioButtonAlist "$curPage"))
            (setq boxesName (gentemp))
            (iht (cons "\\newline\\indent\{5\}\radioboxes{"
              (cons boxesName
                (cons ""
                  (cons "\\htmfile\{pick\}\{\\htmfile\{unpick\}\}" nil)))))
            (setq defaultValue "1")
            (DO ((G166568 buttons (cdr G166568))
              (G166540 nil))
              ((or (atom G166568)
                (progn (setq G166540 (car G166568)) nil)
                (progn
                  (progn
                    (setq message (car G166540))
                    (setq info (cadr G166540))
                    (setq buttonName (caddr G166540))
                    G166540))))

— defun htRadioButtons —
42.2. FUNCTIONS CREATING PAGES

```
nil))
(SEQ (EXIT (progn
  (cond
    ((null (LASSOC buttonName
      (htpInputAreaAlist |$curPage|)))
     (setUpDefault buttonName
       (cons 'button
         (cons defaultValue nil)))
     (setq defaultValue
       "0")))
    (iht (cons "\item{\em\radiobox[" 
        (cons
          (htpLabelDefault |$curPage| buttonName)
          (cons "]"{" 
            (cons buttonText
              (cons "}" 
                (cons boxesName
                  (cons \\\space{} nil)))))))
    (bcIssueHt message)
    (iht "\\space{}")
    (bcIssueHt info))))
  (iht "\\enditems\\indent{0} ")])))
```

defun htBcRadioButtons

— defun htBcRadioButtons —

```
(defun |htBcRadioButtons| (G166594)
  (prog (groupName buttons boxesName message info buttonName defaultValue)
    (declare (special |$curPage|))
    (return
      (SEQ (progn
        (setq groupName (car G166594))
        (setq buttons (cdr G166594))
        (htpSetRadioButtonAlist |$curPage|
          (cons groupName
            (cons (buttonNames buttons)
              (htpRadioButtonAlist |$curPage|)))
        (setq boxesName (gentemp))
        (iht (cons "\\radioboxes{"
          (cons boxesName
            (cons "}{\htbmfile{pick}}{\htbmfile{unpick}} "
```

CHAPTER 42. HYPERDOC BASIC COMMAND SUPPORT

(defun setUpDefault
  (defun setUpDefault (name props)
    (declare (special|$curPage|))
    (|htpAddInputAreaProp||$curPage| name props))

(nil)))
(DO ((G166616 buttons (cdr G166616))
     (G166588 nil))
     ((or (atom G166616)
          (progn (setq G166588 (car G166616)) nil)
          (progn
            (progn
              (setq message (car G166588))
              (setq info (cadr G166588))
              (setq buttonText (caddr G166588))
              G166588)
              nil))
      nil)
  (SEQ (EXIT (progn
      (cond
        ((null (LASSOC buttonText
          (|htpInputAreaAlist||$curPage|)))
          |setUpDefault| buttonText
          (cons 'button
            (cons defaultButtonValue nil)))
          (setq defaultButtonValue
            "0"))
        (iht) (cons
          "{\em\radiobox["
          (cons
            (|htpLabelDefault|$curPage| buttonText)
            (cons "}\{"
              (cons buttonText
                (cons "}\{"
                  (cons boxesName
                    (cons \"nil\)))))))
          (iht) (|bcIssueHt| message)
          (iht) (|bcIssueHt| info))))))
defun buttonNames

— defun buttonNames —

(defun buttonNames (buttons)
  (prog (buttonName)
    (return
      (seq (prog (G166645)
        (setq G166645 nil)
        (return
          (do ((G166651 buttons (cdr G166651))
               (G166637 nil))
            ((or (atom G166651)
                (progn (setq G166651 (car G166651)) nil)
                 (progn
                   (progn
                     (setq buttonName (caddr G166637))
                     G166637
                   nil))
               (reverso G166645))
            (seq (exit (setq G166645
              (cons buttonName G166645))))))))))

defun htInputStrings

— defun htInputStrings —

(defun htInputStrings (strings)
  (prog (mess1 numChars default stringName spadType filter mess2)
    (declare (special $curPage))
    (return
      (seq (progn
        (liht "\newline\indent{5}\beginitems ")
        (do ((G166685 strings (cdr G166685))
             (G166665 nil))
            ((or (atom G166685)
                (progn (setq G166665 (car G166685)) nil)
                 (progn
                   (progn
                     (setq mess1 (car G166665))
                   nil))
            nil)
(setq mess2 (cadr G166665))
(setq numChars (caddr G166665))
(setq default (cadddr G166665))
(setq stringName
  (car (cddddr G166665)))
(setq spadType
  (cdr (cddddr G166665)))
(setq filter (cddr (cddddr G166665)))
G166665)
nil))
nil)
(SEQ (EXIT (progn
  (cond
    ((null (LASSOC stringName
         (htpInputAreaAlist| |$curPage|)))
      (setUpDefault| stringName
        (cons '|string|
          (cons default
            (cons spadType
              (cons filter nil))))))))
  (cond
    ((|htpLabelErrorMsg| |$curPage|
        stringName)
      (iht| (cons "\centerline{{\em 
        (cons
          (|htpLabelErrorMsg| |$curPage|
            stringName)
          (cons "}" nil))))
      (setq mess2
          (CONCAT mess2 (|bcSadFaces|)))
      (|htpSetLabelErrorMsg| |$curPage|
        stringName nil)))
      (iht| "\item ")
      (|bcIssueHt| mess1)
      (iht| (cons "\inputstring{"
        (cons stringName
          (cons ")\{"
            (cons numChars
              (cons ")\{"
                (cons
                  (|htpLabelDefault| |$curPage|
                    stringName)
                (cons "} " nil))))))))
      (|bcIssueHt| mess2))))
      (iht| "\enditems\indent{0}\newline ")))
_______
42.2. FUNCTIONS CREATING PAGES

defun htProcessDomainConditions

— defun htProcessDomainConditions —

(defun |htProcessDomainConditions| (condList)
 (declare (special |$curPage|))
 (|htpSetDomainConditions| |$curPage| (|renamePatternVariables| condList))
 (|htpSetDomainVariableAlist| |$curPage| (|computeDomainVariableAlist|)))

defun renamePatternVariables

— defun renamePatternVariables —

(defun |renamePatternVariables| (condList)
 (declare (special |$curPage| |$PatternVariableList|))
 (progn
   (|htpSetDomainPvarSubstList| |$curPage|
    (|renamePatternVariables1| condList nil
     |$PatternVariableList|))
   (|substFromAlist| condList (|htpDomainPvarSubstList| |$curPage|))))

defun renamePatternVariables1

— defun renamePatternVariables1 —

(defun |renamePatternVariables1| (condList substList patVars)
 (prog (restConds pattern t2 pv t3 cond nsubst)
    (declare (special |$EmptyMode|))
    (return
     (cond
      ((null condList) substList)
      (t (setq cond (car condList))
       (setq restConds (cdr condList))
       (cond
        ((or (and (consp cond) (eq (qcar cond) '|isDomain|)
                (progn
                (setq t2 (qcdr cond)))
                (and (consp t2)

...
```lisp
(defun substFromAlist
  (z substAlist)
  (prog (pvar replace)
    (return
      (SEQ (progn
        (DO ((G166792 substAlist (cdr G166792))
          (G166783 nil))
          ((or (atom G166792)
            (progn (setq G166783 (car G166792)) nil)
            (progn
              (setq pvar (car G166783))
              (setq replace (cdr G166783))
              G166783)
            nil))
        nil)
      (SEQ (EXIT (setq z (subst replace pvar z :test #'equal))))
      z))))
)

(defun computeDomainVariableAlist
  (declare (special |$curPage|))
  (return
    (SEQ (prog (G166813)
      (setq G166813 nil)
      (return
        (DO (((G166819 (|htpDomainPvarSubstList| |$curPage|))
            (cdr G166819))
            (G166805 NIL))
        ((or (atom G166819)
          (progn (setq G166805 (car G166819)) nil)
          (progn
            (setq pvar (cdr G166805))
            G166805)
        NIL))
    ))
  )
)
```
defun pvarCondList

— defun pvarCondList —

(defun pvarCondList (pvar)
  (declare (special $curPage))
  (NREVERSE
    (pvarCondList1 (cons pvar nil) nil
      (htpDomainConditions |$curPage|))))

——

defun pvarCondList1

— defun pvarCondList1 —

(defun pvarCondList1 (pvarList activeConds condList)
  (prog (cond restConds t2 pv t3 pattern)
    (return
      (cond
        ((null condList) activeConds)
        (t (setq cond (car condList))
          (setq restConds (cdr condList))
          (cond
            ((and (consp cond)
                (progn
                  (setq t2 (qcdr cond))
                  (and (consp t2)
                    (progn
                      (setq pv (qcar t2))
                      (setq t3 (qcdr t2))
                      (and (consp t3)
                        (eq (qcdr t3) nil)
                        (progn
                          (setq pattern (qcar t3))
                          ...)))))
            ...)))))

(NREVERSE0 G166813))
(SEQ (EXIT (setq G166813
  (cons (cons pvar
    (pvarCondList1 pvar))
  G166813)))))))

...
42.2. FUNCTIONS CREATING PAGES

(defun pvarsOfPattern)

|— defun pvarsOfPattern —|

(defun pvarsOfPattern (pattern)
  (prog ()
    (declare (special $PatternVariableList))
    (return
     (seq (cond
           ((null (listp pattern)) nil)
           (t
            (prog (G166869)
              (setq G166869 nil)
              (return
               (do ((G166875 (cdr pattern) (cdr G166875))
                    (pvar nil))
                  ((or (atom G166875)
                      (progn (setq pvar (car G166875)) nil))
                   (nreverse0 G166869))
               (seq (exit (cond
                           ((member pvar $PatternVariableList))
                           (setq G166869
                                 (cons pvar G166869)))))
             )))
      ))))

(defun htMakeTemplates,substLabel)

|— defun htMakeTemplates,substLabel —|

(defun htMakeTemplates,substLabel (i template)
  (seq (if (consp template)
         (exit (intern (concat (car template) (princ-to-string i))

```
defun htMakeTemplates

— defun htMakeTemplates —

(defun |htMakeTemplates| (templateList numLabels)
  (prog ()
    (return
     (SEQ (progn
       (setq templateList
         (prog (G166895)
           (setq G166895 nil)
           (return
            (DO ((G166900 templateList
              (CDR G166900))
              (template nil))
              ((or (atom G166900)
                (progn
                 (setq template (car G166900))
                 nil))
              (NREVERSE0 G166895))
            (SEQ (EXIT (setq G166895
              (cons
               ([templateParts| template)
               G166895))))))))

  (prog (G166910)
    (setq G166910 nil)
    (return
     (DO ((i 1 (1+ i))
       ((qsgratep i numLabels)
        (NREVERSE0 G166910))
       (SEQ (EXIT (setq G166910
         (cons
         (prog (G166922)
           (setq G166922 nil)
           (return
            (DO (((G166927 templateList
              (CDR G166927)))
              (template nil))
              ((or (atom G166927)
                (progn
                 (setq template
                  (cdr template)))))))))

  (EXIT template))))))))
defun templateParts

— defun templateParts —

(defun templateParts (template)
  (prog (i)
    (return
      (cond
        ((null (stringp template)) template)
        (t (setq i (SEARCH "%l" template))
          (cond
            ((null i) template)
            (t
              (cons (SUBSEQ template 0 i)
                (SUBSEQ template (+ i 2))))))))))

defun htMakeDoneButton

— defun htMakeDoneButton —

(defun htMakeDoneButton (message func)
  (progn
    (\bcHt| "\newline\vspace{1}\centerline{"
    (cond
      ((equal message "Continue")
        (\bchDoneButton| "\lispdownlink"\ContinueBitmap| func))})))
CHAPTER 42. HYPERDOC BASIC COMMAND SUPPORT

(defun htProcessDoneButton

  (defun |htProcessDoneButton| (arg)
    (let (label func)
      (setq label (car arg))
      (setq func (cadr arg))
      (|ipt| "\newline\vspace{1}\centerline{"
        (cond
          ((equal label "Continue")
            (|ipt||htMakeButton| "\lispdownlink" \ContinueBitmap| func))
          ((equal label "Push to enter names")
            (|ipt||htMakeButton| "\lispdownlink" \ControlBitmap{clicktoset}| func))
          (t
            (|ipt||htMakeButton| "\lispdownlink" (concat "\box{" label "}") | func)))
      (|ipt| "} ")))

(defun htMakeButton

  (defun |htMakeButton| (G166990 &AUX options func message htCommand)
    (DSETQ (htCommand message func . options) G166990)
    (prog (skipStateInfo? id type)
          (declare (special |$curPage|))
          (return
            (SEQ (progn
                (setq skipStateInfo? (ifcar options))
                (|ipt| (cons htCommand (cons |" nil))
                (|bcIssueHt| message)
                (cond
42.2. FUNCTIONS CREATING PAGES

```lisp
{skipStateInfo?
  {iht} (cons "}{(|htDoneButton| '|
    (cons func
      (cons "| "
       (cons (|htpName| |$curPage|)
       (cons ")" nil))))))

{t
  {iht} (cons "}{|htDoneButton| '|
    (cons func
      (cons " | (progn " nil))))
  (DO ((G166977 (|htpInputAreaAlist| |$curPage|)
      (CDR G166977))
    (G166965 nil))
    ((OR (ATOM G166977)
      (progn (setq G166965 (car G166977)) nil)
      (progn
        (setq id (car G166965))
        (setq type (car (cddddr G166965)))
        G166965)
      nil))
    nil)
  (SEQ (EXIT (progn
    {iht} (cons "}{(|htpSetLabelInputString| "
      (cons (|htpName| |$curPage|)
      (cons "'|"
        (cons id
          (cons ")" nil))))))
    (cond
      ((eq type '|string|)
        {iht} (cons "\"\stringvalue{"
          (cons id
            (cons "\"
              nil))))))
    (t
      {iht} (cons "\"\boxvalue{"
        (cons id
          (cons "\"
            nil))))))
    ({iht} "") )))
  (iht} (cons (|htpName| |$curPage|)
    (cons "))" nil))))))))
```
defun bchtMakeButton

— defun bchtMakeButton —

(defun bchtMakeButton (htCommand message func)
  (prog (id type)
    (declare (special |$curPage|))
    (return
      (SEQ (progn
          (|bcHt| (cons htCommand
            (cons "{"
              (cons message
                (cons "}{"(|htDoneButton| ' |"
                  (cons func
                    (cons "|
                      (progn "\nil")))))))
          (DO ((|htpInputAreaAlist| |$curPage|) (cdr |htpInputAreaAlist| |$curPage|)
            (G166992 nil))
            ((or (atom |htpInputAreaAlist|) (eq |htpInputAreaAlist| nil))
              (progn (setq G166992 (car |htpInputAreaAlist|)) nil)
              (progn
                (progn
                  (setq id (car G166992))
                  (setq type (car (cddddr G166992)))
                  (setq id (car (cddddr G166992)))
                  (setq id nil)))))
        (SEQ (EXIT (progn
          (|bcHt| (cons "(|htSetLabelInputString| "
            (cons (|htpName| |$curPage|)
              (cons "|"
                (cons id
                  (cons "]" nil)))))))
          (cond
            ((eq type '|string|)
              (|bcHt| (cons "\\\stringvalue{"
                (cons id
                  (cons "}" nil))))))
            (t
              (|bcHt| (cons "\\\boxvalue{"
                (cons id
                  (cons "}" nil))))))
        )
        (|bcHt| " \nil"))))
      (|bcHt| (cons (|htpName| |$curPage|)
        (cons "}" nil)))))))
    )
  )))
)
42.2. FUNCTIONS CREATING PAGES

---

defun htProcessDoitButton

— defun htProcessDoitButton —

(defun |htProcessDoitButton| (arg)
  (let (label command func fun)
    (setq label (car arg))
    (setq command (cadr arg))
    (setq func (caddr arg))
    (setq fun (|mkCurryFun| func (cons command nil)))
    (|iht| "\newline\vspace{1}\centerline{"
       (|htMakeButton| \lispcommand" (concat \box{" label "}") fun)
    (|iht| "} ")
    (|iht| "\vspace{2}{Select \ \UpButton{} \ to go back one page.}"
     (|iht| "\newline{Select \ \ExitButton{QuitPage} \ to remove this window.}"

---

defun htMakeDoitButton

— defun htMakeDoitButton —

(defun |htMakeDoitButton| (label command)
  (declare (special |$curPage|))
  (progn
    (cond
      ((equal label "Do It")
        (|bcHt| "\newline\vspace{1}\centerline{\lispcommand{\DoItBitmap}{(|doDoitButton| "}}"
      (t
        (|bcHt| (cons "\newline\vspace{1}\centerline{\lispcommand{\box{"
                    (cons label
                      (cons "}){(|doDoitButton| "
                      (nil))})}))
        (|bcHt| (|htpName| |$curPage|))
        (|bcHt| (cons " ""
                    (cons (|htEscapeString| command)
                      (cons "\" nil))))
        (|bcHt| "})"))
        (|bcHt| "\vspace{2}{Select \ \UpButton{} \ to go back one page.}"
        (|bcHt| "\newline{Select \ \ExitButton{QuitPage} \ to remove this window.}")))
defun doDoitButton

— defun doDoitButton —

(defun doDoitButton (htPage command)
  (declare (ignore htPage))
  (executeInterpreterCommand command))

defun executeInterpreterCommand

— defun executeInterpreterCommand —

(defun executeInterpreterCommand (command)
  (progn
    (princ command)
    (terpri)
    (setCurrentLine command)
    (catch 'spad_reader (parseAndInterpret command))
    (princ (mkprompt))
    (finish-output)))

defun htDoneButton

— defun htDoneButton —

(defun htDoneButton (func htPage)
  (cond
    ((typeCheckInputAreas htPage) (htMakeErrorPage htPage))
    ((null (fboundp func))
      (systemError (cons "unknown function" (cons func nil))))
    (t (funcall (symbol-function func) htPage))))
defun typeCheckInputAreas

— defun typeCheckInputAreas —

(defun \texttt{\textbackslash typeCheckInputAreas} \texttt{\textbackslash{htPage}}

(prog \texttt{(inputAlist stringName t2 t3 t4 t5 t6 t7 spadType t8 filter condList string val errorCondition)}

\texttt{(declare (special $\textbackslash bcParseOnly$))}

\texttt{(return}

\texttt{\textbackslash SEQ (progn}

\texttt{(setq inputAlist nil)}

\texttt{(setq errorCondition nil)}

\texttt{(DO ((G167160 (\textbackslash htpInputAreaAlist} \texttt{htPage)}

\texttt{(cdr G167160))}

\texttt{(entry nil)})

\texttt{((or (atom G167160)

\texttt{\textbackslash progn (setq entry (car G167160)) nil)}

\texttt{nil)}

\texttt{\textbackslash SEQ \texttt{\textbackslash EXIT (cond}}

\texttt{((and (consp entry)

\texttt{\textbackslash progn}

\texttt{(setq stringName}

\texttt{(QCAR entry))}

\texttt{(setq t2 (QCDR entry))}

\texttt{(and (consp t2)

\texttt{\textbackslash progn}

\texttt{(setq t3}

\texttt{(QCDR t2))}

\texttt{(and (consp t3)

\texttt{\textbackslash progn}

\texttt{(setq t4}

\texttt{(QCDR t3))}

\texttt{(and (consp t4)

\texttt{\textbackslash progn}

\texttt{(setq t5}

\texttt{(QCDR t4))}

\texttt{(and (consp t5)

\texttt{\textbackslash eq (QCAR t5)

\texttt{\textbackslash 'string\textbackslash )}

\texttt{\textbackslash progn}

\texttt{(setq t6}

\texttt{(QCDR t5))}

\texttt{(and (consp t6)

\texttt{\textbackslash progn}

\texttt{(setq t7}

\texttt{(QCDR t6))}

\texttt{(and

\texttt{(consp t7)

\texttt{\textbackslash progn

\texttt{(setq t8}

\texttt{(QCDR t7))}

\texttt{(and

\texttt{(consp t8)
(setq spadType (QCAR t7))
(setq t8 (QCDR t7))
(and (consp t8) (eq (QCDR t8) nil)
(progn (setq filter (QCAR t8)) t))))))))))))
(progn
(setq condList (LASSOC (LASSOC spadType
(|htpDomainPvarSubstList|
htPage))
(|htpDomainVariableAlist|
htPage)))
(setq string
(|htpLabelFilteredInputString|
htPage stringName))
(cond
(|$bcParseOnly|
(cond
((null (|ncParseFromString| string))
(|htpSetLabelErrorMsg| htPage
"Syntax Error"
"Syntax Error")
(t nil)))
(t
(setq val
(|checkCondition|
(|htpLabelInputString|
htPage stringName)
string condList))
(cond
((stringp val)
(setq errorCondition t)
(|htpSetLabelErrorMsg| htPage
stringName val))
42.2. FUNCTIONS CREATING PAGES

```lisp
(defun checkCondition
  (s1 string condList)
  (prog (pred t2 t3 pvar t4 pattern val type data newType)
    (return
      (cond
        ((and (consp condList) (eq (QCDR condList) nil)
          (progn
            (setq t2 (qcar condList))
            (and (consp t2)
              (eq (QCAR t2) '|Satisfies|)
              (progn
                (setq t3 (QCDR t2))
                (and (consp t3)
                  (progn
                    (setq pvar (QCAR t3))
                    (setq t4 (QCDR t3))
                    (AND (consp t4)
                      (eq (QCDR t4) nil)
                      (progn
                        (setq pred (QCAR t4))
                        (t2)))))))))
      (setq val (funcall pred string))
      (cond
        ((stringp val) val)
        (t (cons '|String| (|wrap| s1)))))
    (null (and (consp condList) (eq (qcdr condList) nil))
      (progn
        (setq t2 (qcar condList))
        (and (consp t2)
          (eq (qcar t2) '|isDomain|)
          (progn
            (setq t3 (QCDR t2))
            (and (consp t3)
              (progn
                (setq pvar (QCAR t3))
                (setq t4 (QCDR t3))
                (setq type (QCDR t4)))
            (cons (|String|) (|wrap| s1)))))))))
```
(and (consp t4)
   (eq (QCDR t4) nil)
   (progn
     (setq pattern
       (QCAR t4))
     t))))))))

(|systemError|
 "currently invalid domain condition")
((equal pattern '(|String|))
 (cons '(|String|) (|wrap| s1)))
(t (setq val (|parseAndEval| string))
 (cond
   ((stringp val)
    (cond
      ((equal val "Syntax Error ")
       "Error: Syntax Error ")
      (t (|condErrorMsg| pattern))))
   (t (setq type (car val))
    (setq data (cdr val))
    (setq newType
      (catch 'spad_reader
        (|resolveTM| type pattern)))
    (cond
      ((null newType) (|condErrorMsg| pattern))
      (t (|coerceInt| val newType)))))
   )))

defun condErrorMsg

---

---

defun condErrorMsg ---

(defun |condErrorMsg| (type)
 (prog (typeString)
   (return
    (progn
      (setq typeString (|form2String| type))
      (cond
       ((consp typeString)
        (setq typeString
          (apply #'CONCAT typeString))
        (CONCAT "Error: Could not make your input into a "
          typeString))))
     (catch 'spad_reader
       (|resolveTM| type pattern)))
    (COND
      ((null newType) (|condErrorMsg| pattern))
      (t (|coerceInt| val newType)))))
   )))

---

defun condErrorMsg

---

defun parseAndEval

— defun parseAndEval —

(defun |parseAndEval| (string)
  (prog (|$InteractiveMode| $boot $spad |$e| |$QuietCommand|)
    (declare (special |$InteractiveMode| $boot $spad |$e|
                 |$QuietCommand|))
    (return
      (progn
        (setq |$InteractiveMode| t)
        (setq $boot nil)
        (setq $spad t)
        (setq |$e| |$InteractiveFrame|)
        (setq |$QuietCommand| t)
        (|parseAndEval1| string))))

———

defun parseAndEval1

— defun parseAndEval1 —

(defun |parseAndEval1| (string)
  (let (v syntaxError pform val)
    (setq syntaxError nil)
    (setq pform
      (progn
        (setq v
          (|applyWithOutputToString| '|ncParseFromString| (cons string nil)))
        (cond
          ((car v) (car v))
          (t (setq syntaxError t) (cdr v))))
    (cond
      (syntaxError "Syntax Error ")
      (pform
        (setq val
          (|applyWithOutputToString| '|processInteractive|
            (cons pform (list nil))))
        (cond
          ((car val) (car val))
          (t "Type Analysis Error"))))
      (t nil))))

———
defun oldParseString

--- defun oldParseString ---

(defun |oldParseString| (string)
  (prog (tree)
    (return
     (progn
       (setq tree
         (|applyWithOutputToString| '|string2SpadTree|
           (cons string nil)))
       (cond
        ((car tree)
         (|parseTransform| (postTransform (car tree))))
        (t (cdr tree)))))))

---

defun makeSpadCommand

--- defun makeSpadCommand ---

(defun |makeSpadCommand| (&rest a1)
  (let (opForm lastArg argList z)
    (setq z a1)
    (setq opForm (concat (car z) "("))
    (setq lastArg (|last| z))
    (setq z (cdr z))
    (setq argList nil)
    (do ((g1 z (cdr g1)) (arg nil))
      ((or (atom g1)
          (progn (setq arg (car g1)) nil)
          (null (nequal arg lastArg)))
     nil)
    (setq argList (cons (concat arg ",") argList))
    (setq argList (nreverse (cons lastArg argList)))
    (concat opForm (apply #'concat argList) ")")))

---

defun htMakeInputList

--- defun htMakeInputList ---
(defun |htMakeInputList| (stringList)
  (prog (lastArg argList)
    (return
      (SEQ (progn
        (setq lastArg (|last| stringList))
        (setq argList nil)
        (DO ((G167328 stringList (cdr G167328)) (arg nil))
          ((or (atom G167328)
              (progn (setq arg (car G167328)) nil)
              (null (NEQUAL arg lastArg)))
            nil)
        (SEQ (EXIT (setq argList
          (cons
            (CONCAT arg ", ")
            argList)))))
        (setq argList (NREVERSE (cons lastArg argList))))
      (|bracketString| (apply #'CONCAT argList))))))

---

defun bracketString

— defun bracketString —

(defun |bracketString| (string)
  (CONCAT "[" string "]")

---

defun quoteString

— defun quoteString —

(defun |quoteString| (string)
  (concat "\"" string "\")

---

defvar $funnyQuote

— initvars —
(defvar |$funnyQuote| \Rubout)

---

defvar $funnyBacks

--- initvars ---

(defvar |$funnyBacks| \200)

---

defun htEscapeString

--- defun htEscapeString ---

(defun |htEscapeString| (str)
  (declare (special |$funnyBacks| |$funnyQuote|))
  (setq str (substitute |$funnyQuote| #" str))
  (substitute |$funnyBacks| #\ str))

---

defun htsv

--- defun htsv ---

(defun |htsv| ()
  (|startHTPage| 50)
  (|htSetVars|))

---

defun htSetVars

--- defun htSetVars ---
42.2. FUNCTIONS CREATING PAGES
(defun |htSetVars| ()
(declare (special |$setOptions| |$lastTree| |$path|))
(setq |$path| nil)
(setq |$lastTree| nil)
(when (nequal 0 (lastatom |$setOptions|)) (|htMarkTree| |$setOptions| 0))
(|htShowSetTree| |$setOptions|))

———-

defun htShowSetTree
— defun htShowSetTree —
(defun |htShowSetTree| (setTree)
(prog (page okList maxWidth1 maxWidth2 tabset1 tabset2 label links)
(declare (special |$path|))
(return
(SEQ (progn
(setq |$path|
(TAKE (- (LASTATOM setTree))
|$path|))
(setq page (|htInitPage| (|mkSetTitle|) nil))
(|htpSetProperty| page ’|setTree| setTree)
(setq links nil)
(setq maxWidth1 (setq maxWidth2 0))
(SEQ (DO ((G167379 setTree (cdr G167379))
(setData nil))
((or (atom G167379)
(progn
(setq setData (car G167379))
nil))
nil)
(SEQ (EXIT (cond
((|satisfiesUserLevel|
(elt setData 2))
(EXIT (progn
(setq okList
(cons setData okList))
(setq maxWidth1
(max
(|#|
(PNAME (elt setData 0)))
maxWidth1))
(setq maxWidth2
(max
(|htShowCount|
(PRINC-TO-STRING

1451


(elt setData 1)))
(maxWidth2))))))))))
(setq maxWidth1 (max 9 maxWidth1))
(setq maxWidth2 (max 41 maxWidth2))
(setq tabset1 (PRINC-TO-STRING maxWidth1))
(setq tabset2
(PRINC-TO-STRING
(-
  (+ maxWidth2 maxWidth1) 1))))
(htSay "|htSay| "\tab{2}\newline Variable\tab{"
(PRINC-TO-STRING
(+ maxWidth1
  (quotient maxWidth2 3))))
"})Description\tab{"
(PRINC-TO-STRING
(+ (+ maxWidth2 maxWidth1) 2))
"})Value\newline\beginitems")
(Do ((G167392 (reverse okList) (CDR G167392))
  (setData nil))
  ((or (atom G167392)
    (progn
      (setData (car G167392))
      nil))
    nil)
  (seq (exit (progn
    (htSay "|htSay| "\item"
    (setq label
      (strconc "\menuitemstyle{" 
        (elt setData 0)
      "}"))
    (setq links
      (cons label
        (cons
          (cons '|text|
            (cons
              "|tab{"
              (cons tabset1
                (cons "}\{\em "
                  (cons
                    (elt setData 1)
                    (cons "|tab{"
                      (cons tabset2
                        (cons "}\{\em "
                          (cons
                            (htShowSetValue
                              setData)
                            (cons
                              "}
                              nil))))))))))))
))))))}}}}}
defun htShowCount

(defun htShowCount (s)
  (prog (m i count)
    (return
     (SEQ (progn
       (setq m (|#| s))
       (cond
        ((> 8 m) (- m 1))
        (t (setq i 0) (setq count 0)
         (DO () ((NULL (> (- m 7) i)) nil)
          (SEQ (EXIT (cond
            ((and (equal (elt s i) #\{)
              (equal (elt s (1+ i)) #\})
              (equal (elt s (+ i 2)) #\e)
              (equal (elt s (+ i 3)) #\m))
            (setq i (1+ i))
            (t (setq i (1+ i))
              (setq count (1+ count))))))))
        (+ count (- m i)))))))))

defun htShowSetTreeValue

(defun htShowSetTreeValue

(defun htShowSetTreeValue (s)
  (return
   (SEQ (progn
     (setq m (|#| s))
     (cond
      ((> 8 m) (- m 1))
      (t (setq i 0) (setq count 0)
       (DO () ((NULL (> (- m 7) i)) nil)
        (SEQ (EXIT (cond
          ((and (equal (elt s i) #\{)
          (equal (elt s (1+ i)) #\})
          (equal (elt s (+ i 2)) #\e)
          (equal (elt s (+ i 3)) #\m))
          (setq i (1+ i))
          (t (setq i (1+ i))
            (setq count (1+ count))))))))
        (+ count (- m i)))))))))
(defun htShowSetTreeValue (setData)
  (let (st)
    (setq st (elt setData 3))
    (cond
      ((eq st 'function)
        (object2String (funcall (elt setData 4) '%display%)))
      ((eq st 'integer)
        (object2String (eval (elt setData 4))))
      ((eq st 'string)
        (object2String (eval (elt setData 4))))
      ((eq st 'literals)
        (object2String (translateTrueFalse2YesNo (eval (elt setData 4))))
        (eq st 'tree) "...")
      (t (systemError)))))

-------

defun mkSetTitle

--- defun mkSetTitle ---

(defun mkSetTitle ()
  (declare (special $path))
  (strconc "Command {\em )set " (listOfStrings2String $path) "}"))

-------

defun listOfStrings2String

--- defun listOfStrings2String ---

(defun listOfStrings2String (u)
  (cond
    ((null u) "")
    (t (strconc (listOfStrings2String (cdr u)) " " (stringize (car u))))))

-------

defun htShowSetPage

--- defun htShowSetPage ---
(defun |htShowSetPage| (htPage branch)
 (let (setTree setData st)
  (declare (special |$path|))
  (setq setTree (|htpProperty| htPage '|setTree|))
  (setq |$path| (cons branch (take (- (lastatom setTree)) |$path|)))
  (setq setData (|assoc| branch setTree))
  (cond
   ((null setData) (|systemError| "No Set Data"))
   (t (setq st (elt setData 3))
     (cond
      ((eq st 'function) (|htShowFunctionPage| htPage setData))
      ((eq st 'integer) (|htShowIntegerPage| htPage setData))
      ((eq st 'literals) (|htShowLiteralsPage| htPage setData))
      ((eq st 'tree) (|htShowSetTree| (elt setData 5)))
      ((eq st 'string) (|htSetNotAvailable| htPage "set compiler")
       (t (|systemError| "Unknown data type")))))))

defun htShowLiteralsPage

— defun htShowLiteralsPage —

(defun |htShowLiteralsPage| (htPage setData)
 (|htSetLiterals| htPage (elt setData 0) (elt setData 1)
  (elt setData 4) (elt setData 5) '|htSetLiteral|)

— defun htSetLiterals —

(defun |htSetLiterals| (htPage name message variable values functionToCall)
  (prog (page links)
    (return
     (SEQ (progn
       (setq page
         (|htInitPage| "Set Command"
          (|htpPropertyList| htPage)))
         (|htsetProperty| page '|variable| variable)
         (|bcHt| (cons "\\centerline{Set \{\em "
            (cons name

42.2. FUNCTIONS CREATING PAGES

---
defun htSetLiteral

— defun htSetLiteral —

(defun |htSetLiteral| (htPage val)
  (|htInitPage| "Set Command" nil)
  (set (|htpProperty| htPage '|variable|) (|translateYesNo2TrueFalse| val))
  (|htKill| htPage val))
defun htShowIntegerPage

(defun htShowIntegerPage (htPage setData)
  (prog (page message t1)
    (declare (special $htFinal $htInitial))
    (return
      (progn
        (setq page
          (htInitPage (mkSetTitle) (htpPropertyList htPage)))
        (htpSetProperty page "$variable (elt setData 4))
        (bcHt (list \"centerline(Set \em " (elt setData 0) \}\newline")
          (setq message (elt setData 1))
        (bcHt (list \{\em Description: } " message \"\newline\vspace{1} "))
          (setq t1 (elt setData 5))
        (setq $htInitial (car t1))
        (setq $htFinal (cadr t1))
        (cond
          ((equal $htFinal (+ $htInitial 1))
            (bcHt "Enter the integer \em ")
          ((bcHt (liststringize $htInitial))
            (bcHt ") or \em ")
          ((bcHt (liststringize $htFinal))
            (bcHt ":")
          ((null $htFinal))
            (bcHt "Enter an integer greater than \em ")
          ((bcHt (liststringize (- $htInitial 1)))
            (bcHt ":")
          (t (bcHt "Enter an integer between \em ")
            (bcHt (liststringize $htInitial))
          ((bcHt ") and \em ")
            (bcHt (liststringize $htFinal))
          ((bcHt ":")
            (htMakePage
              (cons '(domainConditions (|Satisfies| S chkRange))
                (cons (cons 'bcStrings
                   (list (list 5 (eval (elt setData 4)) 'value 'S)))
                 nil)))
          (htSetvarDoneButton "Select to Set Value" 'htSetInteger)
          (htShowPage))))))

        — defun htShowIntegerPage —

(defun htShowIntegerPage (htPage setData)
  (prog (page message t1)
    (declare (special $htFinal $htInitial))
    (return
      (progn
        (setq page
          (htInitPage (mkSetTitle) (htpPropertyList htPage)))
        (htpSetProperty page "$variable (elt setData 4))
        (bcHt (list \"centerline(Set \em " (elt setData 0) \}\newline")
          (setq message (elt setData 1))
        (bcHt (list \{\em Description: } " message \"\newline\vspace{1} "))
          (setq t1 (elt setData 5))
        (setq $htInitial (car t1))
        (setq $htFinal (cadr t1))
        (cond
          ((equal $htFinal (+ $htInitial 1))
            (bcHt "Enter the integer \em ")
          ((bcHt (liststringize $htInitial))
            (bcHt ") or \em ")
          ((bcHt (liststringize $htFinal))
            (bcHt ":")
          ((null $htFinal))
            (bcHt "Enter an integer greater than \em ")
          ((bcHt (liststringize (- $htInitial 1)))
            (bcHt ":")
          (t (bcHt "Enter an integer between \em ")
            (bcHt (liststringize $htInitial))
          ((bcHt ") and \em ")
            (bcHt (liststringize $htFinal))
          ((bcHt ":")
            (htMakePage
              (cons '(domainConditions (|Satisfies| S chkRange))
                (cons (cons 'bcStrings
                   (list (list 5 (eval (elt setData 4)) 'value 'S)))
                 nil)))
          (htSetvarDoneButton "Select to Set Value" 'htSetInteger)
          (htShowPage))))))
defun htSetInteger

-- defun htSetInteger --

(defun |htSetInteger| (htPage)
  (prog (val)
    (return
      (progn
        (|htInitPage| (|mkSetTitle|) nil)
        (setq val
          (|chkRange| (|htpLabelInputString| htPage '|value|)))
        (cond
          ((null (integerp val))
            (|errorPage| htPage
              (cons "Value Error" nil)
            ))
          (t (set (|htpProperty| htPage '|variable|) val)
            (|htKill| htPage val))))))

defun htShowFunctionPage

-- defun htShowFunctionPage --

(defun |htShowFunctionPage| (htPage setData)
  (prog (fn)
    (return
      (cond
        ((setq fn (elt setData 6)) (funcall fn htPage))
        (t (set (|htpSetProperty| htPage '|setData| setData)
            (|htpSetProperty| htPage '|parts| (elt setData 5))
            (|htShowFunctionPageContinued| htPage))))))
defun htShowFunctionPageContinued

— defun htShowFunctionPageContinued —

(defun htShowFunctionPageContinued (htPage)
  (prog (parts setData phrase kind variable checker initValue restParts page currentValue)
    (return
      (progn
        (setq parts (htpProperty htPage 'parts))
        (setq setData (htpProperty htPage 'setData))
        (setq phrase (caar parts))
        (setq kind (cadar parts))
        (setq variable (caddar parts))
        (setq checker (car (cdddar parts)))
        (setq initValue (cadr (cdddar parts)))
        (setq restParts (cdr parts))
        (htpSetProperty htPage 'variable variable)
        (htpSetProperty htPage 'checker checker)
        (htpSetProperty htPage 'parts restParts)
        (cond
          ((eq kind 'literals)
           (htSetLiterals htPage (elt setData 0) phrase variable checker (htFunctionSetLiteral)))
          (t
           (setq page
             (htInitPage (mkSetTitle)
             (htpPropertyList htPage))
             (bcHt (cons "\centerline{Set \em \{\}\newline" nil)))
             (bcHt (cons "{\em Description: } \newline\vspace{1} " nil)))
             (setq currentValue (eval variable))
             (htMakePage)
             (cons (cons 'domainConditions)
             (cons (cons 'Satisfies)
             (cons 'S (cons checker nil)))
             nil))
             (cons (cons 'text phrase)
             (cons (cons 'inputStrings)
             (cons "" nil))
             (cons "" nil)
             (cons 60 nil)
             (cons currentValue nil)
             (cons 'value)
(cons 'S nil)))))
nil))))
(\ltSetvarDoneButton| "Select To Set Value" \htFunCommand|)
(\htShowPage|)))])

---

defun \htSetvarDoneButton

--- defun \htSetvarDoneButton ---

(defun \htSetvarDoneButton| (message func)
  (progn
    (\bcHt| "\\\newline\\vspace{1}\\centerline{"
      (cond
        ((OR (equal message "Select to Set Value")
            (equal message "Select to Set Values"))
          (\bcHt\MakeButton| "\\lisplink"
            "\\\ControlBitmap{clicktoset}" func))
        (t
          (\bcHt\MakeButton| "\\lisplink"
            (CONCAT "\\fbox{" message "}"
            func)))
    (\bcHt| "} "))))

---

defun \htFunctionSetLiteral

--- defun \htFunctionSetLiteral ---

(defun \htFunctionSetLiteral| (htPage val)
  (progn
    (\htInitPage| "Set Command" nil)
    (set (\htProperty| htPage '|'variable|)
      (\translateYesNo2TrueFalse| val))
    (\htFunCommandContinue| htPage val)))
defun htSetFunCommand

--- defun htSetFunCommand ---

(defun |htSetFunCommand| (htPage)
  (let (variable checker value)
    (setq variable (|htpProperty| htPage '|variable|))
    (setq checker (|htpProperty| htPage '|checker|))
    (setq value (|htCheck| checker (|htpLabelInputString| htPage '|value|)))
    (set variable value)
    (|htSetFunCommandContinue| htPage value)))

---

defun htSetFunCommandContinue

--- defun htSetFunCommandContinue ---

(defun |htSetFunCommandContinue| (htPage value)
  (let (parts continue)
    (setq parts (|htpProperty| htPage '|parts|))
    (setq continue
      (cond
        ((null parts) nil)
        (and (consp parts)
          (consp (qcar parts)) (eq (qcaar parts) '|break|)
          (consp (qcddar parts)) (eq (qcddar parts) nil))
          (eval (qcdr parts)))
        (t t)))
    (cond
      (continue
        (|htpSetProperty| htPage '|parts| (qcdr parts))
        (|htShowFunctionPageContinued| htPage))
      (t (|htKill| htPage value))))

---

defun htKill

--- defun htKill ---

(defun |htKill| (htPage value)
(declare (ignore htPage))
(progn (string)
(declare (special |$path|))
(return
(progn
  (htInitPage| "System Command" nil)
  (setq string
    (STRENC "\{\em }set \\
    (|listOfStrings2String| (cons value |$path|))")
  (htMakePage| ((|text| . ")\vspace{1}\newline\rm"))
  (htProcessDoitButton|
    (cons "Press to Remove Page"
      (cons "" (cons '|htDoNothing| nil)))
  (htShowPage|)))))

(defun htSetNotAvailable
  (htPage whatToType)
(let page string
  (setq page
    (htInitPage| "Unavailable Set Command" (|htpPropertyList| htPage)))
  (htInitPage| "Unavailable System Command" nil)
  (setq string (strconc "{\em " whatToType "\}"))
  (htMakePage|
    (cons '{|text| ")\vspace{1}\newline"
      "{Sorry, but this system command is not available through HyperDoc. Please directly issue this\"\vspace{2}\newline\centerline{\tt"
        (cons (cons '|text| string) nil)))
  (htProcessDoitButton|
    (list "Press to Remove Page"
      "" (cons '|htDoNothing| nil)))
  (htShowPage|)))

--------

defun htSetNotAvailable

  — defun htSetNotAvailable —

(defun |htSetNotAvailable| (htPage whatToType)
  (let page string
    (setq page
      (htInitPage| "Unavailable Set Command" (|htpPropertyList| htPage)))
    (htInitPage| "Unavailable System Command" nil)
    (setq string (strconc "{\em " whatToType "\}"))
    (htMakePage|
      (cons '{|text| ")\vspace{1}\newline"
        "{Sorry, but this system command is not available through HyperDoc. Please directly issue this\"\vspace{2}\newline\centerline{\tt"
          (cons (cons '|text| string) nil)))
    (htProcessDoitButton|
      (list "Press to Remove Page"
        "" (cons '|htDoNothing| nil)))
    (htShowPage|)))

--------
defun htDoNothing

— defun htDoNothing —

(defun htDoNothing (htPage command)
  (declare (ignore htPage command))
  nil)

—

defun htCheck

— defun htCheck —

(defun htCheck (checker value)
  (cond
    ((consp checker) (htCheckList checker (parseWord value)))
    (t (funcall checker value))))

—

defun parseWord

— defun parseWord —

(defun parseWord (x)
  (prog ()
    (return
     (SEQ (cond
       ((stringp x)
        (cond
         ((prog (G167588)
           (setq G167588 t)
           (return
            (DO ((G167594 nil (null G167588))
                 (G167595 (maxindex x))
                 (i 0 (1+ i)))
             ((OR G167594 (QSGREATERP i G167595))
              G167588))
            (QSGREATERP i G167595)
            G167588))
         (QSGREATERP i G167595)
         (AND G167588
          (digitp (elt x i))))))))))

        )
    )))
  )
  )
)
defun htCheckList

— defun htCheckList —

(defun htCheckList (checker value)
  (prog (n t2 m)
    (return
     (progn
      (cond
        ((member value '(y ye yes Y YE YES)) (setq value 'yes))
        (cond
          ((member value '(n no N NO)) (setq value 'no))
          (cond
            ((and (consp checker)
                (progn
                  (setq n (qcar checker))
                  (setq t2 (qcdr checker))
                  (and (consp t2) (eq (qcdr t2) nil)
                    (progn (setq m (qcar t2)) t)))))
            (integerp n))
            (cond
              ((eql m (1+ n))
               (cond ((member value checker) value) (t n)))
              (null m)
              (cond
                ((and (integerp value) (>= value n)) value)
                (t n)))))
            ((integerp m)
             (cond
              ((and (integerp value) (>= value n) (<= value m)) value)
              (t n)))))
            (member value checker)
            (t (car checker)))))))
defun translateYesNoToTrueFalse

— defun translateYesNoToTrueFalse —

(defun translateYesNoToTrueFalse (x)
  (cond
    ((eq x 'yes) t)
    ((eq x 'no) nil)
    (t x)))

defun chkNameList

— defun chkNameList —

(defun chkNameList (x)
  (prog (u parsedNames)
    (return
      (SEQ (progn
        (setq u ('bcString2ListWords x))
        (setq parsedNames
          (prog (G167635)
            (setq G167635 nil)
            (return
              (DO ((G167640 u (CDR G167640))
                (x nil))
                ((or (atom G167640)
                  (progn
                    (setq x (car G167640))
                    nil))
                (NREVERSE0 G167635))
              (SEQ (EXIT (setq G167635
                (cons (|ncParseFromString| x)
                  G167635))))))))

    (cond
      ((prog (G167646)
        (setq G167646 t)
        (return
          (DO ((G167652 nil (NULL G167646))
            (G167653 parsedNames (CDR G167653))
            (x nil))
            ((OR G167652 (ATOM G167653)
              (progn (setq x (car G167653)) nil))
            G167646))))
CHAPTER 42. HYPERDOC BASIC COMMAND SUPPORT

```lisp
(SEQ (EXIT (setq G167646
               (AND G167646 (identp x))))))
parsedNames)
(t
 "Please enter a list of identifiers separated by blanks")))))

---

defun chkPosInteger

---

(defun chkPosInteger (s)
  (prog (u)
    (return
     (cond
      ((and (setq u (parseOnly s)) (integerp u) (> u 0))
       u)
      (t "Please enter a positive integer"))))

---

defun chkOutputFileName

---

(defun chkOutputFileName (s)
  (cond
   ((member (bcString2WordList s) '(CONSOLE |console|))
    'console)
   (t (chkDirectory s))))

---

defun chkDirectory

---

(defun chkDirectory (s) s)
```
defun chkNonNegativeInteger

— defun chkNonNegativeInteger —

(defun |chkNonNegativeInteger| (s)
  (prog (u)
    (return
     (cond
      ((and (setq u (|ncParseFromString| s)) (integerp u)
       (>= u 0))
        u)
     (t "Please enter a non-negative integer")))))))

———

defun chkRange

— defun chkRange —

(defun |chkRange| (s)
  (prog (u)
    (declare (special |$htFinal| |$htInitial|)))
    (return
     (cond
      ((and (setq u (|ncParseFromString| s)) (integerp u)
        (>= u |$htInitial|)
        (or (null |$htFinal|) (<= u |$htFinal|)))
        u)
      (null |$htFinal|)
        (STRCONC "Please enter an integer greater than "
          (|stringize| (- |$htInitial| 1))))
     (t
        (STRCONC "Please enter an integer between 
          (|stringize| |$htInitial|) " and 
          (|stringize| |$htFinal|)))))))

———

defun chkAllNonNegativeInteger

— defun chkAllNonNegativeInteger —
(defun \chkAllNonNegativeInteger\ (s)
  (prog (u)
    (return
     (or (and (setq u (\ncParseFromString\ s))
           (\member\ u '(\a\ al\ all\ A AL ALL)) 'ALL)
          (\chkNonNegativeInteger\ s)
           "Please enter {\em all} or a non-negative integer"))))

---

defun htMakePathKey,fn

---

defun htMarkTree
(rplacd (last tree) n)
(SEQ (DO ((G167706 tree (cdr G167706)) (branch nil))
     ((OR (ATOM G167706)
          (progn (setq branch (car G167706)) nil))
      nil)
  (SEQ (EXIT (cond
                     ((eq (elt branch 3) 'tree)
                      (EXIT ([htMarkTree| (elt branch 5)
                              (1+ n))))))))))

|----------------|
defun htSetHistory |
|----------------|

(defun [htSetHistory| (htPage)
  (let (msg data)
    (setq msg
      '|when the history facility is on (yes), results of computations are saved in memory|)
    (setq data
      (list '|history| msg '|history| 'literals '|$HiFiAccess|
        '(|on| |off| |yes| |no|)))
    ([htShowLiteralsPage| htPage data])))

|----------------|
defun htSetOutputLibrary |
|----------------|

(defun [htSetOutputLibrary| (htPage)
  ([htSetNotAvailable| htPage ")set compiler output")

|----------------|
defun htSetInputLibrary |
|----------------|
(defun |htSetInputLibrary| (htPage)
  (|htSetNotAvailable| htPage " set compiler input"))

---

defun htSetExpose

— defun htSetExpose —

(defun |htSetExpose| (htPage)
  (|htSetNotAvailable| htPage " set expose"))

---

defun htSetOutputCharacters

— defun htSetOutputCharacters —

(defun |htSetOutputCharacters| (htPage)
  (|htSetNotAvailable| htPage " set output characters"))

---

defun htSetLinkerArgs

— defun htSetLinkerArgs —

(defun |htSetLinkerArgs| (htPage)
  (|htSetNotAvailable| htPage " set fortran calling linker"))

---

defun htSetCache

— defun htSetCache —
42.2. FUNCTIONS CREATING PAGES

(defun htSetCache (&REST arg &AUX options htPage)
  (declare (special $valueList $path))
  (setq htPage (car arg))
  (setq options (cdr arg))
  (setq $path '([functions |cache|])
  (setq htPage ([htInitPage ([mkSetTitle] nil)]
  (setq $valueList nil)
  ([htMakePage]
    '((text
        "Use this system command to cause the AXIOM interpreter to 'remember' 
        past values of interpreter functions."
        "To remember a past value of a function, the interpreter "
        "sets up a \{em cache\} for that function based on argument values."
        "When a value is cached for a given argument value, its value is gotten 
        from the cache and not recomputed. Caching can often save much "
        "computing time, particularly with recursive functions or functions that "
        "are expensive to compute and that are called repeatedly "
        "with the same argument." 
        "\vspace{1}\newline"
      )
        (domainConditions ([Satisfies] $ chkNameList))
    )
    (text
      "Enter below a list of interpreter functions you would like specially cached."
      "Use the name {\em all} to give a default setting for all "
      "interpreter functions."
      "Enter \{\em all\} or a list of names (separate names by blanks):"
      ([inputStrings] ("" "" 60 "all" names S))
    )
    ([doneButton] "Push to enter names" [htCacheAddChoice])))
  )
[htShowPage]))

defun htCacheAddChoice

— defun htCacheAddChoice —

(defun [htCacheAddChoice] (htPage)
  (prog (names page)
    (declare (special $valueList)))
    (return
      (SEQ (progn
        (setq names
          ([bCtString2WordList]
            ([htpLabelInputString] htPage 'names)))
        (setq $valueList)
          (cons ([listOfStrings2String] names)
            $valueList))
        (cond
          ((null names) ([htCacheAddQuery])))
    )
)
 CHAPTER 42. HYPERDOC BASIC COMMAND SUPPORT

((null (cdr names)) (htCacheOne names))
(t (setq page (htInitPage (mkSetTitle) nil))
 (htpSetProperty page 'names names)
 (htMakePage)
 ’(((domainConditions
   (Satisfies ALLPI chkAllPositiveInteger))
   (|text|
 "For each function, enter below a {\em cache length}, a positive integer. "
 "This number tells how many past values will "
 "be cached."
 "A cache length of {\em 0} means the function won’t be cached."
 "To cache all past values, "
 "enter {\em all}."
 "\vspace{1}\newline"
 "For each function name, enter {\em all} or a positive integer:")))
 (DO ((i 1 (QSADD1 i))
 (G167755 names (CDR G167755)) name nil)
 (or (atom G167755)
 (progn (setq name (car G167755)) nil)
 nil)
 (SEQ (EXIT (htMakePage)
   (cons (cons 'inputStrings)
     (cons
      (cons
       (STRCONC "Function {\em "
       name "} will cache")
       "values"
       (cons 5
       (cons 10
       (cons
        (htMakeLabel
        "c" i)
        (cons 'ALLPI nil))))))
     nil)
   (htSetvarDoneButton "Select to Set Values" 'htCacheSet)
 (htShowPage))))))))

---

defun htMakeLabel

— defun htMakeLabel —

(defun |htMakeLabel| (prefix i)
 (intern (strconc prefix (|stringize| i))))
defun htCacheSet

— defun htCacheSet —

(defun htCacheSet (htPage)
  (prog (names num n name val)
    (declare (special "$cacheCount" "$cacheAlist"))
    (return 
      (SEQ (progn
        (setq names (htpProperty htPage 'names))
        (DO ((i 1 (QSADD1 i))
          (G167785 names (CDR G167785)) (name nil))
          ((or (atom G167785)
              (progn (setq name (car G167785)) nil))
            nil)
        (SEQ (EXIT (progn
          (setq num
            (chkAllNonNegativeInteger
              (htpLabelInputString htPage
                (htMakeLabel "c" i))))
          (setq "$cacheAlist"
            (ADDASSOC (intern name) num "$cacheAlist")))))
      (cond
        ((setq n (LASSOC 'all "$cacheAlist"))
          (setq "$cacheCount" n)
          (setq "$cacheAlist"
            (deleteAssoc 'all "$cacheAlist")))
        (htInitPage "Cache Summary" nil)
        (bcHt "In general, interpreter functions ")
        (bcHt (cond
          ((EQL "$cacheCount" 0)
            "will {\em not} be cached.")
          (t (bcHt "cache ")
            (htAllOrNum "$cacheCount"
              "} values.")
            (bcHt "\vspace{1}\newline ")(bcHt "$cacheAlist"
            (DO ((G167801 "$cacheAlist" (cdr G167801))
              (G167774 nil))
              (or (atom G167801)
                (progn


defun htAllOrNum

— defun htAllOrNum —

(defun |htAllOrNum| (val)
  (cond
    ((eq val '|all|) "\em all")
    ((eql val 0) "\em no")
    (t
      (strconc "the last \em \
               (\stringize\ val)))))

---

defun htCacheOne

— defun htCacheOne —

(defun |htCacheOne| (names)
(prog (page)
  (return
    (setq page (htInitPage (mkSetTitle) nil))
    (htSetProperty page '|names| names)
    (htMakePage
     '((domainConditions|
          ((Satisfies ALLPI |chkAllPositiveInteger|))
          (text| "Enter below a \{em cache length\}, a positive integer. "
            "This number tells how many past values will "
            "be cached. To cache all past values, "
            "enter \{em all\}. " |vspace{1}\newline")
          (inputStrings| |c1| ALLPI)
          ("Enter \{em all\} or a positive integer:" "5 10 |
           |c1| ALLPI))))
     (htSetvarDoneButton |Select to Set Value|
      'htCacheSet))
     (htShowPage))))

-----

defvar $historyDisplayWidth

— initvars —

(defun $historyDisplayWidth 120)

-----

defvar $newline

— initvars —

(defun $newline #\Newline)

-----

defun downlink

— defun downlink —
(defun downlink (page)
  (htInitPage "Bridge" nil)
  (htSay "\replacepage{" page "}")
  (htShowPage))

----------

(defun dbNonEmptyPattern
  (defun dbNonEmptyPattern)

(defun dbNonEmptyPattern (pattern)
  (cond
   ((null pattern) "*")
   (t (setq pattern (PRINC-TO-STRING pattern))
      (cond ((> (|#| pattern) 0) pattern) (t "+"))))

----------

(defun htSystemVariables,g
  (defun htSystemVariables,g)

(defun htSystemVariables,g (t1 al)
  (let (class key options)
    (declare (special |$heading| |$levels|))
    (setq class (caddr t1))
    (setq key (cadddr t1))
    (setq options (cadr (cddddr t1)))
    (cond
     ((null (member class |$levels|)) al)
     ((or (or (eq key 'literals) (eq key 'integer))
          (eq key 'string))
      (cons (cons |$heading| t1) al))
     ((eq key 'tree)
      (htSystemVariables,fn options al nil))
     ((eq key 'function)
      (cons (cons |$heading| t1) al))
     (t (|systemError| key))))

----------
42.2. FUNCTIONS CREATING PAGES

(defun htSystemVariables, fn

    — defun htSystemVariables, fn —

(defun htSystemVariables, fn (t1 al firstTime)
  (declare (special $heading))
  (SEQ (if (atom t1) (EXIT al))
    (if firstTime (setq $heading (opOf (car t1))) nil)
    (EXIT (htSystemVariables, fn (cdr t1)
       (htSystemVariables, gn (car t1) al) firstTime))))

———

defun htSystemVariables, displayOptions

    — defun htSystemVariables, displayOptions —

(defun htSystemVariables, displayOptions (name class variable val options)
  (SEQ (if (eq class 'integer)
    (EXIT (SEQ (htMakePage
      (cons (cons 'bcLispLinks
        (cons
          (cons 'text
            (cons (elt options 0)
              (cons "="
                (cons (or (elt options 1)
                  ""
                    nil)))))
          nil)
        ""
      (cons
        (htSetSystemVariableKind
          (cons variable
            (cons name
              (cons 'parse-integer nil)))))
        nil))))
    (htMakePage
      '((domainConditions
          (isDomain INT (Integer)))))
    (htSystemVariables, displayOptions (name class variable val options)))
  nil)
  (htSystemVariables, displayOptions (name class variable val options))))
defun htSystemVariables, functionTail

— defun htSystemVariables, functionTail —

(defun |htSystemVariables, functionTail| (name class var valuesOrFunction)
  (prog (val)
    (return
     (SEQ (setq val (|eval| var)))
     (if (atom valuesOrFunction)
       (EXIT (SEQ (|htMakePage|
           '(((|domainConditions|
             (|isDomain| STR (|String|)))))
           (|htMakePage|
             (cons (cons '|bcLinks|
               (cons
                (cons (cons variable
                  (cons x nil))
               nil))))
           nil))))
     nil)))))))

---

(defun htSystemVariables, functionTail

  (defun |htMakePage|
    (list
     (cons '|bcStrings| (list (list 5 (princ-to-string val) name 'int)))))
  (if (eq class 'string)
    (EXIT (|htSay| "{\em " val
      "}\space{1}"))
    (EXIT (DO ((G167913 options (cdr G167913)) (x nil))
      ((or (atom G167913)
          (progn (setq x (car G167913)) nil))
     nil)
      (SEQ (if (or (or (equal val x)
        (and (eq val t)
          (eq x '|on|))
        (and (null val) (eq x '|off|)))
        (EXIT (|htSay| "{\em x
          "}\space{1}"))
      (EXIT (|htMakePage|
        (cons (cons '|bcLispLinks|
          (cons (cons x
            (cons " "
              (cons '|htSetSystemVariable|
                (cons (cons variable
                  (cons x nil))
               nil))))
          nil))))
      nil)))))))

---
defun htSystemVariables

— defun htSystemVariables —

(defun htSystemVariables ()
  (prog (|$levels| |$heading| classlevel table heading name
t1 msg key variable options func lastHeading
  (DECLARE (SPECIAL |$levels| |$heading| |$setOptions| |$UserLevel|
     |$fullScreenSysVars|))
  (return
    (SEQ (cond
          ((null |$fullScreenSysVars|) (|htSetVars|))
          (t (setq classlevel |$UserLevel|)
             (setq |$levels| '(|compiler| |development| |interpreter|))
             (setq |$heading| nil)
             (DO () ((NULL (NEQUAL classlevel (car |$levels|))) nil)
               (SEQ (EXIT (setq |$levels| (cdr |$levels|)))
             (setq table
               (NREVERSE
               (|htSystemVariables,fn| |$setOptions| nil
               t)))
     (|htInitPage| "System Variables" nil))

---
(\beginmenu)
(setq lastHeading nil)
(\DO ((G167961 table (cdr G167961)) (G167879 nil))
  ((or (atom G167961)
       (progn (setq G167879 (car G167961)) nil)
       (progn
         (progn
           (setq heading (car G167879))
           (setq name (cadr G167879))
           (setq message (caddr G167879))
           (setq key (car (cddddr G167879)))
           (setq variable (cadr (cddddr G167879)))
           (setq options (caddr (cddddr G167879)))
           (setq func (cadddr (cddddr G167879)))
           G167879)
         nil))
  nil)
(\SEQ (EXIT (progn
    (\beginitem)
    (\cond
      ((equal heading lastHeading)
        (\beginitem)
        t
        (\beginitem)
        (setq heading
          (\beginitem)
        (setq lastHeading heading)))
      (\beginitem)
      (beginitem)
      (\beginitem)
      (beginitem)
      (\beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beginitem)
      (beg
\begin{verbatim}
(DO
  ((G167971 (cdr options)
    (cdr G167971))
    (option nil))
  ((or (atom G167971)
    (progn
      (setq option (car G167971))
      nil))
    nil)
  (SEQ
    (EXIT
      (cond
        ((and (consp option)
          (eq (QCAR option)
            '|break|))
          '|skip|)
        (t
          (setq msg (car option))
          (setq class (cadr option))
          (setq var (caddr option))
          (setq valuesOrFunction (cadddr option))
          (|htSay| "\newline\tab{22}" msg
            "\tab{80}"
            (|htSystemVariables,functionTail| name class var valuesOrFunction))))))
      (t (setq val (|eval| variable))
        (|htSystemVariables,displayOptions| name key variable val options))))))
|htShowPage|))))
\end{verbatim}

defun htSetSystemVariableKind

— defun htSetSystemVariableKind —

(defun htSetSystemVariableKind (htPage arg)
  (let (variable name fun value)
    (setq variable (car arg))
    (setq name (cadr arg))
    (setq fun (caddr arg))
    (setq value (|htpLabelInputString| htPage name))
    (when (and (stringp value) fun) (setq value (funcall fun value)))
    (set variable value))
defun htSetSystemVariable

    (defun htSetSystemVariable (htPage arg)
        (declare (ignore htPage))
        (let (name value)
            (setq name (car arg))
            (setq value (cadr arg))
            (setq value
                (cond
                    ((eq value '|on|) t)
                    ((eq value '|off|) nil)
                    (t value)))
            (set name value)
            (htSystemVariables)))

defun htGloss

    (defun htGloss (pattern)
        (htGlossPage nil
            (or ((dbNonEmptyPattern pattern) "*") t))

defun htGlossPage

    (defun htGlossPage (htPage pattern tryAgain?)
        (prog (|$wildCard| |$key| filter grepForm results defstream
            lines heading k tick)
(declare (special |$wildCard| |$key| |$tick|))
(return (SEQ (progn
   (setq |$wildCard| #\*)
   (cond
      ((equal pattern "**")
       (|downlink| '|GlossaryPage|))
      (t (setq filter (|pmTransFilter| pattern))
       (setq grepForm (|mkGrepPattern| filter '|none|))
       (setq |$key| '|none|)
       (setq results (|applyGrep| grepForm '|gloss|))
       (setq defstream
            (make-instream
             (STRCONC (getenviron "AXIOM"
                          "/algebra/glossdef.text")))
       (setq lines
            (|gatherGlossLines| results defstream))
       (setq heading
            (cond
              ((equal pattern "")
               "Glossary")
              ((null lines)
               (cons "No glossary items match \{\em "
                    (cons pattern
                    (cons "}" nil)))
              (t
               (cons "Glossary items matching \{\em "
                    (cons pattern
                    (cons "}" nil))))))))
   (cond
      ((null lines)
       (cond
        ((and tryAgain? (> (|#| pattern) 0))
         (cond
          ((equal
            (elt pattern (MAXINDEX pattern)))
           #\s)
          ([htGlossPage| htPage
            (SUBSTRING pattern 0 k) t])
          ((upper-case-p (elt pattern 0))
           ([htGlossPage| htPage (downcase pattern) nil])
          (t
           ([errorPage| htPage
            (cons "Sorry"
            (cons nil
            (cons \"\centerline{"
             (append heading
             (append results)))))))))
        (t
         (append heading
         (append results))))))
)
(cons "}" nil)))

(t (|errorPage| htPage
   (cons "Sorry"
     (cons nil
       (cons
         (cons "\centerline{"
           (append heading
            (cons "}" nil)))
         nil)))))))

(t (|htInitPageNoScroll| nil heading)
 (|htSay| "\beginscroll\beginmenu")
 (DO ((G168058 lines (cdr G168058))
      (line nil))
   ((or (atom G168058)
        (progn (setq line (car G168058)) nil))
    nil)
   (SEQ (EXIT (progn
                 (setq tick
                   (|charPosition| |$tick|
                    line 1))
                 (|htSay|
                  "\item{\em \menuitemstyle{}}\tab{0}{\em "
                   (|escapeString|
                    (SUBSTRING line 0 tick))
                  "} "
                   (SUBSTRING line
                    (1+ tick) nil))))))

( |htSay| "\endmenu ")
( |htSay| "\endscroll\newline ")
( |htMakePage|
   (cons (cons '|bcLinks|
      (cons
        (cons "Search"
         (cons ""
          (cons '|htGlossSearch|
            (cons nil nil))))
         nil))
   (|htSay| " for glossary entry matching ")
 (|htMakePage|
   (cons (cons '|bcStrings|
      (cons
        (cons 24
         (cons "*"
          (cons '|filter| (cons 'em nil))))
        nil))
   nil))}
defun gatherGlossLines

(defun gatherGlossLines (results defstream)
  (prog (n keyAndTick byteAddress line k pointer x
    j nextPointer xtralines acc)
    (declare (special $tick))
    (return
      (SEQ (progn
        (setq acc nil)
        (DO ((G168098 results (cdr G168098))
            (keyline nil))
        ((or (atom G168098)
            (progn (setq keyline (car G168098)) nil))
          nil)
        (SEQ (EXIT (progn
          (setq n
            (|charPosition| $tick keyline 0))
          (setq keyAndTick
            (SUBSTRING keyline 0
              (1+ n)))
          (setq byteAddress
            (|string2Integer|
              (SUBSTRING keyline (1+ n)
                nil))))
          (file-position defstream byteAddress)
          (setq line (readline defstream))
          (setq k
            (|charPosition| $tick line 1))
          (setq pointer
            (SUBSTRING line 0 k))
          (setq def
            (SUBSTRING line (1+ k)
              nil))
          (setq xtralines nil)
          (DO ()
            ((null (and (null (eop defstream))
              (setq x
                (readline defstream))
              (setq j
                (|charPosition| $tick x 1))))))
defun htGlossSearch

— defun htGlossSearch —

(defun |htGlossSearch| (htPage junk)
  (declare (ignore junk))
  (|htGloss| (|htLabelInputString| htPage '|filter|)))
defun htGreekSearch

(defun htGreekSearch (filter)
  (prog (ss s names matches nonmatches)
    (return
      (SEQ (progn
        (setq ss (dbNonEmptyPattern filter))
        (setq s (pmTransFilter ss))
        (cond
          ((and (consp s) (eq (QCAR s) 'error))
            (bcErrorPage s))
          ((null s)
            (errorPage nil
              (cons (cons "Missing search string" nil)
                (cons nil
                  (cons "\vspace{2}\centerline{To select one of the greek letters:}\newline "
                    (cons "\centerline{\em first} enter a search key into the input area}\newline "
                      (cons "\centerline{\em then } move the mouse cursor to the work {\em search} and click}"
                          nil)))))))
          (t (setq filter (patternCheck s))
            (setq names
              '(\alpha \beta \gamma \delta \epsilon \zeta \eta \theta \iota \kappa \lambda \mu \nu \pi))
            (DO ((G168149 names (CDR G168149)) (x nil))
                (or (atom G168149)
                  (progn (setq x (car G168149)) nil))
                nil)
            (cond
              (((superMatch? filter (PNAME x))
                (setq matches
                  (cons x matches)))
              (t
                (setq nonmatches
                  (cons x nonmatches))))))
      (setq matches (nreverse matches))
      (setq nonmatches (nreverse nonmatches))
      (htInitPage "Greek Names" nil)
      (cond
        ((null matches)
          (htInitPage)
            (cons "Greek names matching search string {\em "
              (cons ss (cons "}" nil)))))
CHAPTER 42. HYPERDOC BASIC COMMAND SUPPORT

nil)
(|htSay| '|vspace{2}|\centerline{Sorry, but no greek letters match your search string}|centerline{|htShowPage|})
(t
(|htInitPage|
 (cons "Greek letters matching search string {\em "
  (cons ss (cons "}" nil)))
 nil)
(cond
 (nonmatches
  (|htSay| "The greek letters that {\em match} your search string {\em "
  ss "}:")
  (t
   (|htSay| "Your search string {\em "
  ss "} matches all of the greek letters:|)
   (|htSay| "{\em \table{"
   (DO ((G168158 matches (CDR G168158))
         (x nil))
     ((or (atom G168158)
         (progn (setq x (car G168158)) nil))
       nil)
     (SEQ (EXIT (|htSay| "{ x 
                  "}"))))
   (|htSay| "}}\vspace{1}"
   (cond
    (nonmatches
     (|htSay| "The greek letters that {\em do not match} your search string:{\em \table{")
     (DO ((G168167 nonmatches (CDR G168167))
           (x nil))
   ((or (atom G168167)
       (progn
        (setq x (car G168167))
       nil))
     (SEQ (EXIT (|htSay| "{ x 
                  "}"))))
     (|htSay| "}}\vspace{1}
   (|htShowPage|))))))


defun htTextSearch

— defun htTextSearch —

(defun |htTextSearch| (filter)
  (prog (s lines matches nonmatches)
    (return
     (SEQ (progn
           (setq s
                (|pmTransFilter| (|dbNonEmptyPattern| filter)))
           (cond
            ((and (consp s) (eq (QCAR s) '|error|))
             (|bcErrorPage| s))
            ((null s)
             (|errorPage| nil
              (cons (cons "Missing search string"
                     nil)
               (cons nil
                 (cons
                  "\vspace{2}\centerline{To select one of the lines of text:}\newline"
                  (cons
                   "\centerline{{\em first} enter a search key into the input area}\newline"
                   (cons
                    "\centerline{{\em then } move the mouse cursor to the work {\em search} and click}"
                    nil))))))))
            (t (setq filter s)
               (setq lines
               (cons
                "{{\em Fruit flies} *like* a {{\em banana and califlower ears.}}"
                (cons
                 "{{\em Sneak Sears Silas with Savings Snatch}}"
                 nil)))
               (DO ((G168191 lines (cdr G168191)) (x nil))
                 ((or (atom G168191)
                      (progn (setq x (car G168191)) nil))
                  nil)
               (SEQ (EXIT (cond
                            (||superMatch?| filter x)
                            (setq matches
                               (cons x matches)))
                               (t
                                (setq nonmatches
                                   (cons x nonmatches)))))))
      (setq matches (NREVERSE matches))
      (setq nonmatches (NREVERSE nonmatches))
      (|htInitPage| "Text Matches" nil)
      (cond
       ((null matches)
        (|htInitPage|)
CHAPTER 42. HYPERDOC BASIC COMMAND SUPPORT

(cons "Lines matching search string {\em "
      (cons s (cons "}" nil)))
   nil)
   (|htSay|
    '\\vspace{2}\centerline{Sorry, but no lines match your search string}
    '\\vspace{2}\centerline{Click on the up-arrow to try again}}))
   (|htShowPage|))
(\t
 (|htInitPage|
   (cons "Lines matching search string {\em "
      (cons s (cons "}" nil)))
   nil)
   (cond
    (nonmatches
     (|htSay| "The lines that {\em match} your search string {\em 
             s ":")}))
   (t
    (|htSay| "Your search string {\em "
            s "} matches both lines:\})
    (|htSay| "{\em \table{"
    (DO ((G168200 matches (CDR G168200))
      (x nil))
      ((or (atom G168200)
         (progn (setq x (car G168200)) nil))
      nil)
      (SEQ (EXIT (|htSay| "{ x
                    "}\}))
    (|htSay| 
     "}\\vspace{1}"
    ))
   (cond
    (nonmatches
     (|htSay| "The line that {\em does not match} your search string:{\em \table{")
     (DO ((G168209 nonmatches (cdr G168209))
       (x nil))
      ((or (atom G168209)
         (progn
          (setq x (car G168209))
         nil))
      nil)
      (SEQ (EXIT (|htSay| "{ x
                    "}\}))
    (|htSay| 
     "}\})\\vspace{1}"
    ))
   (|htShowPage|)))))))
defun htTutorialSearch

--- defun htTutorialSearch ---

(defun htTutorialSearch (pattern)
  (prog (s source target lines t1 name title)
    (return
      (SEQ (progn
        (setq s
          (or (|dbNonEmptyPattern| pattern)
            (return
              (|errorPage| nil
                (cons "Empty search key"
                  (cons nil
                    (cons
                      "\vspace{3}\centerline{You must enter some search string" nil))))))))
        (setq s (|mkUnixPattern| s))
        (setq source "$AXIOM/doc/hypertex/pages/ht.db")
        (setq target "/tmp/temp.text.$SPADNUM")
        (OBEY (STRCONC "$AXIOM/lib/hthits"
          " " s " "
          source " > " target))
        (setq lines (|dbReadLines| '|temp|))
        (|htInitPageNoScroll| nil
          (cons "Tutorial Pages mentioning \{em "
            (cons pattern (cons ")" nil))))
        (|htSay| "\beginscroll\table{"
          (DO ((G168241 lines (cdr G168241)) (line nil))
            (or (atom G168241)
              (progn (setq line (car G168241)) nil)
                nil)
            (SEQ (EXIT (progn
              (setq t1 (|dbParts| line 3 0))
              (setq name (car t1))
              (setq title (cdr t1))
              (|htSay| (cons "\{downlink{"
                (cons title
                  (cons "}" nil)
                    (cons name
                      (cons "}" nil))))))))
              (|htSay| "})")
            (|htShowPage|)))))))

---
defun mkUnixPattern

— defun mkUnixPattern —

(defun |mkUnixPattern| (s)
  (prog (starPositions k u)
    (declare (special |$wild|))
    (return
      (SEQ (progn
        (setq u (|mkUpDownPattern| s))
        (setq starPositions
          (reverse (prog (G168264)
            (setq G168264 nil)
            (return
              (DO
                ((G168270
                  (+ (- 1)
                    (MAXINDEX u)))
                  (i 1 (QSADD1 i)))
                ((QSGREATERP i G168270)
                  (NREVERSE0 G168264))
                (SEQ
                  (EXIT
                    (cond
                      ((equal (elt u i)
                        |$wild|)
                        (setq G168264
                          (cons i G168264))))))))
            (DO ((G168277 starPositions (cdr G168277))
                (i nil))
              ((or (atom G168277)
                (progn (setq i (car G168277)) nil))
                nil)
              (SEQ (EXIT (setq u
                (STRCONC (SUBSTRING u 0 i)
                  ".*"
                  (SUBSTRING u (1+ i) nil))))))
            (cond
              ((NEQUAL (elt u 0) |$wild|)
                (setq u (STRCONC "[^a-zA-Z]" u)))
              (t (setq u (SUBSTRING u 1 nil)))))
            (cond
              ((NEQUAL (elt u (setq k (MAXINDEX u))) |$wild|)
                (setq u (STRCONC u "[^a-zA-Z-Z]")))
              (t (setq u (SUBSTRING u 0 k))))
          u))))
      ))))
    )))
  )))
))
Chapter 43

Browser Support Code

43.1 Pages Initiated from HyperDoc Pages

Search routines

defun dKind

    — defun dbKind 0 —

(defun |dbKind| (line)
  (elt line 0))

—

defun checkFilter

[trimString p??]

    — defun checkFilter —

(defun |checkFilter| (filter)
  (setq filter (princ-to-string filter))
  (if (string= filter "")
     "*"
     (|trimString| filter)))

—
CHAPTER 43. BROWSER SUPPORT CODE

;concatWithBlanks r == ; r is [head,:tail] =⇒ ; tail =⇒ STRCONC(head," ",concatWithBlanks tail) ; head ; ""

defun Concatenate words with blanks

— defun concatWithBlanks 0 —

(defun |concatWithBlanks| (r)
  (if (consp r)
    (format nil "~{~a~^ ~}" r)
    "")

———

defun Make constructor names lowercase

[ifcar p ??]
[hget p 1194]
[conLowerCaseConTran p 1494]
[$lowerCaseConTb p ??]

— defun conLowerCaseConTran —

(defun |conLowerCaseConTran| (x)
  (declare (special |$lowerCaseConTb|))
  (cond
    ((identp x) (or (ifcar (hget |$lowerCaseConTb| x)) x))
    ((atom x) x)
    (t (loop for y in x collect (|conLowerCaseConTran| y))))

———

defun string2Constructor

[downcase p 1206]
[hget p 1194]
[ifcar p ??]
[$lowerCaseConTb p ??]

— defun string2Constructor —

(defun |string2Constructor| (x)
(declare (special \$lowerCaseConTb\)))
(cond
 ((null (stringp x)) x)
 (t (or (ifcar (hget \$lowerCaseConTb\ (intern (downcase x)))) x))))

---

defvar dbDelimiters

— initvars —

(defvar \$dbDelimiters\ (list #\space #\( #\)) )

---

defun String to words respecting delimiters

This breaks a string into words respecting delimiters, so if

\$dbDelimiters = ( #\space #\( #\))

then

(\$dbString2Words\ "now is (the) time")
Value = ("now" "is" #\( "the" #\) "time")

---

defun dbString2Words —

(defun \$dbString2Words\ (z)
 (loop
   with i = 0
   with pair = nil
   do (setq pair (\$dbWordFrom\ z i))
   while (and (consp pair) (= (length pair) 2)) ; dbWordFrom(1,i) is [v,i]]
   do (setq i (second pair))
   collect (first pair)))

---
defun Next word respecting delimiters

This returns the next word or the next delimiter. So given

```
$dbDelimiters = ( #\space #( #\) )

(defun dbWordFrom| "now is (the) time")
```

```
(dbWordFrom| b 0) Value = ("now" 3)
(dbWordFrom| b 3) Value = ("is" 6)
(dbWordFrom| b 6) Value = (#( 8)
(dbWordFrom| b 8) Value = ("the" 11)
(dbWordFrom| b 11) Value = (#\) 12)
(dbWordFrom| b 12) Value = ("time" 17)
(dbWordFrom| b 17) Value = NIL
```

This creates a page for any cat, dom, package, default package
constructors Cname\$\#\$E\$sig \$args \$abb \$comments (C is C, D, P, X)

There are 8 parts of an htPage:

1. kind
2. name
3. nargs
4. xflag
5. sig
6. args
7. abbrev
8. comments

[dbXParts p??]
[mkConform p1532]
[opOf p??]
[capitalize p??]
[ncParseFromStream p1240]
[dbSourceFile p??]
[dbConformGenUnder p??]
[strconc p??]
[isExposedConstructor p1026]
[htInitPageNoScroll p??]
[htAddHeading p1415]
[htSayStandard p??]
[htpSetProperty p1408]
[dbShowConsDoc1 p1550]
[addParameterTemplates p??]
[htSay p??]
[htSayStandard p??]
[kPageContextMenu p??]
[htShowPageNoScroll p1416]
[$atLeastOneUnexposed p??]
[$conformsAreDomains p??]

— defun kPage —

(defun kPage (a1)
  (let (parts name nargs sig args form isFile kind
         conform conname capitalKind signature sourceFileName constrings
         emString heading page options line)
    (declare (special $conformsAreDomains $atLeastOneUnexposed)))
(setq line (car a1))
(setq options (cdr a1))
;; constructors Cname\#E\sig \args \abb \comments (C is C, D, P, X)
(setq parts (|dbXParts| line 7 1))
(setq kind (first parts))
(setq name (second parts))
(setq nargs (third parts))
(setq sig (fifth parts))
(setq form (ifcar options))
(setq isFile (null kind))
(setq kind (or kind "package"))
(rplaca parts kind)
(setq conform (|mkConform| kind name args))
(setq conname (|opOf| conform))
(setq capitalKind (|capitalize| kind))
(setq signature (|ncParseFromString| sig))
(setq sourceFileName (|dbSourceFile| (intern name)))
(setq constrings
  (if (ifcdr form)
    (|dbConformGenUnder| form)
    (list (strconc name args))))
(setq emString (cons "{\sf " (append constrings (list "}"))))
(setq heading (cons capitalKind (cons " " emString)))
(unless (|isExposedConstructor| conname)
  (setq heading (cons "Unexposed " heading)))
(setq page (|htInitPageNoScroll| NIL))
(|htAddHeading| heading)
(|htSayStandard| '|\beginscroll |)
(|htpSetProperty| page '|isFile| t)
(|htpSetProperty| page '|parts| parts)
(|htpSetProperty| page '|heading| heading)
(|htpSetProperty| page '|kind| kind)
(|htpSetProperty| page '|conform| conform)
(|htpSetProperty| page '|signature| signature)
; what follows is stuff from kiPage with domain = nil
(setq |$conformsAreDomains| nil)
(|dbShowConsDoc1| page conform nil)
(when (and (nequal kind '|category|) (> nargs 0))
  (|addParameterTemplates| page conform))
(when |$atLeastOneUnexposed|
  (|htSay| "\newline{}{\em *} = unexposed")
(|htSayStandard| '|\endscroll |)
(|kPageContextMenu| page)
(|htShowPageNoScroll|)))
defun Hyperdoc category search
[constructorSearch p1501]

---
defun cSearch ---

(defun cSearch (filter)
  (constructorSearch (checkFilter filter) 'c "category")

---
defun Hyperdoc default domain search
[constructorSearch p1501]

---
defun xSearch ---

(defun xSearch (filter)
  (constructorSearch (checkFilter filter) 'x "default package")

---
defun Hyperdoc domain search
[constructorSearch p1501]

---
defun dSearch ---

(defun dSearch (filter)
  (constructorSearch (checkFilter filter) 'd "domain")

---
defun Hyperdoc package search
[constructorSearch p1501]

---
defun pSearch ---

(defun pSearch (filter)
  (constructorSearch (checkFilter filter) 'p "package")

---
defun Hyperdoc constructor search
[constructorSearch p1501]

   — defun kSearch —

(defun |kSearch| (filter)
   (|constructorSearch| (|checkFilter| filter) '|k| "constructor")

   ——

defun Hyperdoc default constructor search
[constructorSearch p1501]

   — defun ySearch —

(defun |ySearch| (filter)
   (|constructorSearch| (|checkFilter| filter) '|y| "constructor")

   ——

defun Read libdb.text at file-position n

   — defun dbRead 0 —

(defun |dbRead| (n)
   (with-open-file
      (instream (strconc (getenviron "AXIOM") "/algebra/libdb.text")))
      (file-position instream n)
      (read-line instream)))

   ——

defun String trim with newlines removed

   — defun libdbTrim 0 —

(defun |libdbTrim| (s)
   (string-trim '('#\space #\tab #\newline) (substitute #'\space #'\newline s)))

   ——
defun Hyperdoc common constructor search

(defun constructorSearch (filter key kind)
  (let (parse pageName name u line newkind page message)
    (declare (special \$lowerCaseConTb))
    (cond
      ((null filter) nil)
      ((setq parse (conSpecialString? filter)) (conPage parse))
      (t
        (setq name (if (stringp filter) (intern filter) filter))
        (when (setq u (hget \$lowerCaseConTb name))
          (setq filter (princ-to-string (car u))))
        (cond
          ((setq line (conPageFastPath (downcase filter)))
            (setq newkind (case (dbKind line)
              (#\p "package")
              (#\d "domain")
              (#\c "category")))
            (cond
              (or (equal kind "constructor") (equal kind newkind))
              (\kPage line))
          (t
(setq page (|htInitPage| "Query Page" nil))
(|htSetProperty| page ’|line| line)
(setq message
 (list "{\em " (|dbName| line) "} is not a {\em " kind
 "} but a {\em " newkind
 "}. Would you like to view it?\vspace{1} " ))
(|htQuery| message ’|grepConstructorSearch| 't)
(|htShowPage|)))
((equal filter "*")
 (|grepSearchQuery| kind
 (list filter key kind ’|constructorSearchGrep| )))
(t (|constructorSearchGrep| filter key kind)))))))

---

defun conSpecialString?

[ifcar p??]
[string2Words p??]
[ncParseFromString p1240]
[member p1198]
[conLowerCaseConTran p1494]
[contained p??]
[kisValidType p1530]
[strconc p??]
[dbString2Words p1495]
[string2Constructor p1494]
[conSpecialString? p1502]

— defun conSpecialString? —

(defun |conSpecialString?| (&REST a1 &AUX options filter)
 (let (secondTime t1 words parse form u)
 (setq filter (car a1))
 (setq options (cdr a1))
 (setq secondtime (ifcar options))
 (setq t1 (|string2Words| filter))
 (setq parse
 (cond
 ((and (consp t1) (not (qcdr t1))) ; t1 is [s]
  (setq words (|ncParseFromString| (qcar t1))))
 ((every #'(lambda (x) (null (|member| x ’("and" "or" "not")))) words)
  (|ncParseFromString| filter))))
 (cond
 ((null parse) nil)
 (t
 (setq form (|conLowerCaseConTran| parse)))
)
43.1. PAGES INITIATED FROM HYPERDOC PAGES

(defun conPage (a &rest b)
  (let ((|$conArgstrings| form da pageName line)
        (declare (special |$conArgstrings|))
        ; the next 4 lines allow e.g. MATRIX INT ==> Matrix Integer (see kPage)
        (setq form (if (atom a) (cons a b) a))
        (setq |$conArgstrings| (loop for x in (ifcdr a) collect (|form2HtString| x)))
        (cond ((null (atom a)) (setq a (car a))))
        (setq da (downcase a))
        (cond
          ((setq pageName
               (lassq da
                '((|type| . |CategoryType|)
                  (|union| . |DomainUnion|)
                  (|record| . |DomainRecord|)
                  (|mapping| . |DomainMapping|)
                  (|enumeration| . |DomainEnumeration|)))))
        (setq line (|conPageFastPath| da)) ; lower case name of cons?
  ))
(setq line (|conPageFastPath| (upcase a))) ; upper case an abbrev?
(|kPage| line form))
(t (|ySearch| a))) ; slow search (include default packages)

---

defun gets line quickly for constructor name or abbreviation

[|length p??|]
[|charPosition p??|]
[|lassq p??|]
[|dbRead p1500|]
[|conPageConEntry p1504|]
[|$lowerCaseConTb p??|]

---

(defun |conPageFastPath| (x)
  (let (s name entry lineNumber)
    (declare (special |$lowerCaseConTb|))
    (setq s (princ-to-string x))
    (unless (> (|#| s) (|charPosition| #\* s 0)) ; quit if name has * in it
      (setq name (cond ((stringp x) (intern x)) (t x)))))
    (when entry
      ;;'dbLineNumbers property is set by function dbAugmentConstructorDataTable
      (if (setq lineNumber (lassq '|dbLineNumber| (cddr entry)))
        (|dbRead| lineNumber)
        (|conPageConEntry| (car entry)))))

---

defun conPageConEntry

[|conPageConEntry buildLIdbConEntry (vol9)|]
[|$conName p??|]
[|$conform p??|]
[|$exposed? p??|]
[|$doc p??|]
[|$kind p??|]
43.1. PAGES INITIATED FROM HYPERDOC PAGES

(defun |conPageConEntry| (entry)
  (let ([|$conname|] [|$conform|] [|$exposed?|] [|$doc|] [|$kind|])
    (declare (special |$conname| [|$conform|] [|$exposed?|] [|$doc|] [|$kind|]))
    (setq |$conname| nil)
    (setq [|$conform|] nil)
    (setq [|$exposed?|] nil)
    (setq [|$doc|] nil)
    (setq [|$kind|] nil)
    ([|buildLibdbConEntry| entry]))

defun kdPageInfo

[|htSay| p?]
[nequal p?]
[bcHt p1414]
[|htSayStandard| p?]
[kPageArgs p?]
[length p?]
[extractFileNameFromPath p?]
[subseq p?]
[getdatabase p1156]
[|htSay| p?]
[|htMakePage| p1416]

— defun kdPageInfo —

(defun |kdPageInfo| (name abbrev nargs conform signature file?)
  (let (sourceFileName filename)
    ([|htSay|] '{|\sf | name "}"
    (when (nequal abbrev name) ([|bcHt|] (list '} has abbreviation | abbrev))
    (when file? ([|bcHt|] (list " is a source file.")))
    (cond
      ((eql nargs 0)
       (when (nequal abbrev name) ([|bcHt|] ".")))
      (t
       ([|bcHt|]
        (if (eql nargs 1)
            " takes one argument:"
            (list '} takes | (princ-to-string nargs) '| arguments:'))))))
  ([|htSayStandard|] "{indentrel\{2}"
  (when (> nargs 0) ([|kPageArgs|] conform signature))
  ([|htSayStandard|] "{indentrel\{-2}"
  (when (char= (elt name 1- (|| name))) \&
    (setq name (subseq name 0 1- (|| name))))
(setq sourceFileName (getdatabase (intern name) 'sourcefile))
(setq filename (extractFileNameFromPath sourceFileName))
(when (nequal filename "")
  (htSayStandard "\newline{}")
  (htSay "The source code for the constructor is found in ")
  (htMakePage)
  (list (list 'text "\unixcommand{" filename "}{AXIOM/lib/SPADE\ edit "
           sourceFileName " " name "}")))  
(when (nequal nargs 0) (htSay ".")))

---

defun kArgPage

(htpProperty p1408)
(getConstructorModemap p??)
(position p??)
(sublisFormal p??)
(mkDomTypeForm p1507)
(domainDescendantsOf p1507)
(htpSetProperty p1408)
(dbShowCons p1545)

--- defun kArgPage ---

(defun |kArgPage| (htPage arg)
  (let (conform op args domname source n typeForm domTypeForm descendants rank)
    (setq conform (htpProperty htPage '|conform|))
    (setq op (car conform))
    (setq args (cdr conform))
    (setq domname (htpProperty htPage '|domname|))
    (setq source (cddar (getConstructorModemap op)))
    (setq n (position arg args))
    (setq typeForm (sublisFormal args (elt source n)))
    (setq domTypeForm (mkDomTypeForm typeForm conform domname))
    (setq descendants (domainDescendantsOf typeForm domTypeForm))
    (htSetProperty htPage '|cAlist| descendants)
    (setq rank
      (unless (> n 4) (elt (|First| |Second| |Third| |Fourth| |Fifth|) n)))
    (htSetProperty htPage '|rank| rank)
    (htSetProperty htPage '|thing| "argument")
    (dbShowCons htPage '|names|)))

---
defun mkDomTypeForm

[sublislis p??]
[mkDomTypeForm p1507]
[hasIndent p??]

— defun mkDomTypeForm —

(defun mkDomTypeForm (typeForm conform domname)
  (cond
    (domname (sublislis (cdr domname) (cdr conform) typeForm))
    ((and (consp typeForm) (eq (qcar typeForm) '|Join|))
      (cons '|Join|
        (loop for t1 in (qcdr typeForm) collect
          (|mkDomTypeForm| t1 conform domname))))
    ((null (|hasIdent| typeForm)) typeForm)))

defun domainDescendantsOf

[systemError p??]
[simpHasPred p??]
[quickAnd p??]
[domainsOf p??]
[ifcdr p??]
[qcar p??]
[qcdr p??]
[assoc p??]
[listSort p??]
[function p??]
[delete p??]

— defun domainDescendantsOf —

(defun domainDescendantsOf (conform domform)
  (labels ((catScreen (r alist)
              (let (t1 item pred pred1 npred)
                (dolist (x r)
                  (unless (and (consp x) (member (qcar x) '(attribute signature)))
                    (|systemError| x))
                  (setq alist
                    (dolist (anitem alist (nreverse0 t1))
                      (setq item (car anitem))
                      (setq pred (cdr anitem)))))))
(when (and
  (setq pred1 (|simpHasPred| (list '|has| item x)))
  (setq npred (|quickAnd| pred1 pred))
  (setq t1 (cons (cons item npred) t1)))))))

;j keep only those domains that appear in ALL parts of Join
(jfn (arg domlist)
  (let (y r item pred u keepList alist)
    (setq y (car arg))
    (setq r (cdr arg))
    (setq alist (|domainsOf| y (ifcar domlist)))
    (dolist (x r)
      (setq domlist (ifcdr domlist))
      (when (and (consp x) (eq (qcar x) 'category) (consp (qcdr x)))
        (setq alist (catScreen (cddr x) alist)))
      (setq keepList nil)
      (dolist (dom (|domainsOf| x (ifcar domlist)))
        (setq item (car dom))
        (setq pred (cdr dom))
        (when (setq u (|assoc| item alist))
          (setq keepList
            (cons (cons item (|quickAnd| (cdr u) pred)) keepList))))
      (setq alist keepList))
    (dolist (pair alist)
      (rplacd pair (|simpHasPred| (cdr pair))))
    (|listSort| #'glesseqp alist))))

(if (consp conform)
  (cond
    ((eq (qcar conform) '|Join|)
      (jfn
        (|delete| '((|Type| object) (qcdr conform))
          (|delete| '((|Type| object) (ifcdr domform))))
        (eq (qcar conform) 'category) nil)
      (t (|domainsOf| (conform domform))))
      (|domainsOf| (conform domform))))

——

43.2 Branches of Constructor Page

defun kiPage

[htpProperty p1408]
[mkConform p1532]
[kDomainName p1528]
[errorPage p??]
--- defun kiPage ---

(defun kiPage (htPage junk)
 (declare (ignore junk))
 (let ((lt1 kind name nargs args conform domname heading page)
     (declare (special $conformsAreDomains))
     (setq lt1 (htpProperty htPage '|parts|))
     (setq kind (first lt1))
     (setq name (second lt1))
     (setq nargs (third lt1))
     (setq args (sixth lt1))
     (setq conform (mkConform kind name args))
     (setq domname (kDomainName htPage kind name nargs))
     (cond
      ((and (consp domname) (eq (qcar domname) '|error|))
       (errorPage htPage domname))
      (t
       (setq heading
            (list "Description of " (capitalize kind) " {\sf " name args "}")
            (setq page (htInitPage heading (htCopyProplist htPage)))
            (setq $conformsAreDomains domname)
            (ldbShowConsDoc1 htPage conform nil)
            (htShowPage))))))

---

defun kePage
(defun kePage (htPage junk)
  (declare (ignore junk))
  (let ((|$conformsAreDomains| lt1 kind name nargs args constring domname conform conname heading data conlist attrlist oplist prefix page)
    (declare (special |$conformsAreDomains|))
    (setq lt1 (|htpProperty| htPage '|parts|))
    (setq kind (first lt1))
    (setq name (second lt1))
    (setq nargs (third lt1))
    (setq args (sixth lt1))
    (setq constring (concat name args))
    (setq domname (|kDomainName| htPage kind name nargs))
    (cond
      ((and (consp domname) (eq (qcar domname) '|error|))
       (|errorPage| htPage domname))
      (t
       (|httpSetProperty| htPage '|domname| domname)
       (setq |$conformsAreDomains| domname)
       (setq conform (|mkConform| kind name args))
       (setq conname (|opOf| conform))
       (setq heading
         (list (|capitalize| kind) " {\sf ")
           (if domname (|form2HtString| domname nil t) constring "}")
        ))
    (setq data
      (|sublisFormal|
       (or (ifcdr domname) (cdr conform))
       (|getConstructorExports| (or domname conform) t))))
    (setq conlist (car data))
    )
)
\begin{menu}
\begin{tab}{2}
All attributes and operations from:
\end{tab}
\end{menu}
(setq u (assoc op opAlist))
(setq opAlist
  (insertAlist1 (zeroOneConvert op)
    (cons (list sig pred) u)
    opAlist)))

---

defun kePageDisplay

[defun kePageDisplay]

(defun kePageDisplay (htPage which opAlist)
  (let (count total expandProperty data)
    (setq count (length opAlist))
    (cond
      ((eql count 0) nil)
      (t
       (setq total
         (apply #+ (loop for entry in opAlist collect (length (cdr entry))))))
      (if (string= which "operation")
        (httpSetProperty htPage '|opAlist| opAlist)
        (httpSetProperty htPage '|attrAlist| opAlist))
      (setq expandProperty
        (if (string= which "operation")
          'expandOperations|
          'expandAttributes|))
      (httpSetProperty htPage expandProperty '|lists|)
      (httpMakePage
        (list
          (list '|bcLinks| (list (menuButton) "" '|dbShowOps| which '|names|)))))
    (unless (= count total)
      (if (eql count 1)
        (httpSay "1 name for ")
        (httpSay (princ-to-string count) " names for ")))
(if (> total 1)
  (|htSay| (princ-to-string total) " " (|pluralize| which) " are explicitly exported:"))
  (|htSay| "1 " which " is explicitly exported:"
(setq data (|dbGatherData| htPage opAlist which '|names|))
  (|dbShowOpItems| which data nil))))

defun ksPage
[htpProperty p1408]
[kDomainName p1528]
[errorPage p??]
[form2HtString p??]
[htpSetProperty p1408]
[htInitPageNoScroll p??]
[htCopyPropList p??]
[htSay p??]
[htSayStandard p??]
[dbSearchOrder p1514]
[dbShowCons p1545]

— defun ksPage —

(defun |ksPage| (htPage junk)
  (declare (ignore junk))
  (let ((lt1 kind name nargs domname heading domain conform page u)
        (setq lt1 (|htpProperty| htPage '|parts|))
        (setq kind (first lt1))
        (setq name (second lt1))
        (setq nargs (third lt1))
        (setq domname (unless (string= kind "category") (eval domname)))
        (cond
         ((and (consp domname) (eq (qcar domname) '|error|))
          (|errorPage| htPage domname))
         (t
          (setq heading
            (if (null domname)
              (|htpProperty| htPage '|heading|)
              (list "{\sf " (|form2HtString| domname nil t) "}")) )
          (when domname
            (|htpSetProperty| htPage '|domain| domname)
            (|htpSetProperty| htPage '|heading| heading))
          (setq domain (unless (string= kind "category") (eval domname)))
          (setq conform (|htpProperty| htPage '|conform|))
          (setq page
defun dbSearchOrder

[opOf p??]
[dbInfovec p??]
[getdatabase p156]
[simpCatPredicate p??]
[sublisis p??]
[kTestPred p1543]
[devaluate p??]
[kFormatSlotDomain p??]
[dbSubConform p1544]
[dbAddChain p1545]
[$domain p??]
[$infovec p??]
[$predvec p??]

— defun dbSearchOrder —

(defun |dbSearchOrder| (conform domname |$domain|)
  (declare (special |$domain|))
  (let ((|$infovec| name u catpredvec catinfo catvec p pred pak catform res
catforms t1)
    (declare (special |$infovec| |$predvec|))
    (setq conform (or domname conform))
    (setq name (|opOf| conform))
    (setq |$infovec| (|dbInfovec| name))
    (when |$infovec|
      (setq u (elt |$infovec| 3))
      (setq |$predvec|
        (if |$domain| (elt |$domain| 3) (getdatabase name 'predicates)))
      (setq catpredvec (car u))
      (setq catinfo (cadr u))
      (setq catvec (caddr u)))

(defun \|htInitPageNoScroll\| (\|htCopyProplist\| htPage)
  (cons "Search order for " heading)))
(hSay (concat
  "When an operation is not defined by the domain, the following "
  "domains are searched in order for a \"default definition\")
  \|htSayStandard\| "\beginscroll ")
(setq u (|dbSearchOrder| conform domname domain))
(\|htpSetProperty\| htPage '|cAlist| u)
(\|htpSetProperty\| htPage '|thing| "constructor")
(|dbShowCons| htPage '|names|)))
(setq catforms
  (dotimes (i (maxindex catvec) (nreverse0 t1))
    (cond
      ((progn
          (setq pred
            (|simpCatPredicate|
              (progn
                (setq p
                  (sublislis (cdr conform) |$FormalMapVariableList|
                    (|kTestPred| (elt catpredvec i))))
                (if |$domain| (eval p) p))))
          (when (and domname (contained '$ pred))
            (setq pred (subst domname '$ pred :test #'equal)))
          (and (setq pak (elt catinfo i)) pred))
        (setq t1
          (cons
            (cond
              ((and pak (null (identp pak)))
                (|devaluate| pak))
              (t
                (setq catform (|kFormatSlotDomain| (elt catvec i)))
                (setq res (|dbSubConform| (cdr conform)
                  (cons pak (cons '$ (cdr catform))))
                  (when domname (setq res (subst domname '$ res :test #’equal)))
                  res))
              pred)
            t1))))))
  (append (|dbAddChain| conform) catforms))))

------

defun kcPage

[htarProperty p1408]
[kDomainName p1528]
[qcar p??]
[errorPage p??]
[opOf p??]
[form2HtString p??]
[htarInitPage p1415]
[htarCopyProplist p??]
[htarSetProperty p1408]
[dbpHasDefaultCategory? p??]
[htar p??]
[brCon p??]
[htarSayStandard p??]
defun kcPage (htPage junk)
(declare (ignore junk))
(let (lt1 kind name narg xpart domname conform conname heading page message)
(declare (special \defaultPackageNamesHT))
(setq lt1 (htpProperty htPage '\parts))
(setq kind (first lt1))
(setq name (second lt1))
(setq narg (third lt1))
(setq xpart (fourth lt1))
(setq domname (\domainName htPage kind name narg))
(cond
((and (consp domname) (eq (qcar domname) '\error))
 (errorPage htPage domname))
(t
 (setq conform (htpProperty htPage '\conform))
 (setq conname (\opOf conform))
 (setq heading
  (if (null domname)
    (htpProperty htPage '\heading)
    (list "\sf " (\form2HtString domname nil t) "")))
 (setq page
  (htInitPage (cons "Cross Reference for " heading)
    (\htCopyProplist htPage)))
 (when domname
  (htSetProperty htPage '\domname domname)
  (htSetProperty htPage '\heading heading))
 (when (and (string= kind "category")
            (dbpHasDefaultCategory? xpart))
  (htSay "This category has default package ")
  (bcCon (concat name \\& ""))
  (\htSayStandard "\newline"
   (\htBeginMenu 3)
   (\htSayStandard "\item ")
  (setq message
    (if (string= kind "category")
    (list "Categories it directly extends")
    (list "Categories the ")
    )
  )
)
(if (string= kind "default package") "package" kind)
"belongs to by assertion"))))
(|htMakePage|
(list
 (list '|bcLinks|
   (list "\\menuitemstyle{Parents}"
     (list (list '|text| "\\tab{12}" message)) '|kcpPage| nil))))
(|satBreak|)
(setq message
 (if (string= kind "category")
   (list "All categories it is an extension of")
   (list "All categories the " kind " belongs to")))
(|htMakePage|
(list
 (list '|bcLinks|
   (list "\\menuitemstyle{Ancestors}"
     (list (list '|text| "\\tab{12}" message)) '|kcaPage| nil))))
(when (string= kind "category")
 (|satBreak|)
(|htMakePage|
(list
 (list '|bcLinks|
   (list "\\menuitemstyle{Children}"
     (list (list '|text| "\\tab{12}" "Categories which directly extend this category")))))
(|satBreak|)
(|htMakePage|
(list
 (list '|bcLinks|
   (list "\\menuitemstyle{Descendants}"
     (list (list '|text| "\\tab{12}" "All categories which extend this category")))))
(|satBreak|)
(setq message "Constructors mentioning this as an argument type")
(|htMakePage|
(list
 (list '|bcLinks|
   (list "\\menuitemstyle{Dependents}"
     (list (list '|text| "\\tab{12}" message)) '|kcdePage| nil))))
(when (nequal kind "category")
 (|satBreak|)
(|htMakePage|
(list
 (list '|bcLinks|
   (list "\\menuitemstyle{Lineage}"
     "\\tab{12}Constructor hierarchy used for operation lookup"
     '|ksPage| nil))))
(when (string= kind "category")
 (|satBreak|)
(|htMakePage|)
defun kcpPage

[htpProperty p1408]
[kDomainName p1528]
[errorPage p??]
[qcar p??]
[form2HtString p??]
defun kcpPage
  (htPage junk)
  (declare (ignore junk))
  (let ((lt1 kind name nargs domname heading conform conname page parents choice))
    (setq lt1 (htpProperty htPage 'parts))
    (setq kind (first lt1))
    (setq name (second lt1))
    (setq nargs (third lt1))
    (setq domname (extractDomainName htPage kind name nargs))
    (cond
      ((and (consp domname) (eq (qcar domname) 'error))
       (errorPage htPage domname))
      (t
       (setq heading
         (if (null domname)
           (htpProperty htPage 'heading)
           (list '{\sf " (form2HtString domname nil t) "``"})))
       (when domname
        (htpSetProperty htPage 'domname domname)
        (htpSetProperty htPage 'heading heading)
        (setq conform (htpProperty htPage 'conform))
        (setq conname (opOf conform))
        (setq page
          (htInitPage (cons "Parents of " heading) (htCopyProplist htPage)))
        (setq parents (parentsOf conname))
        (when domname
         (setq parents (sublislis (cdr domname) (cdr conform) parents)))
        (htpSetProperty htPage 'cAlist parents)
        (htpSetProperty htPage 'thing "parent")
        (setq choice (if domname 'parameters 'names))
        (dbShowCons htPage choice))))))

defun reduceAlistForDomain
  (sublislis p??)
  (simpHasPred p??)
--- defun reduceAlistForDomain ---

(defun reduceAlistForDomain (alist domform conform)
  (let (pred result)
    (setq alist (sublislis (cdr domform) (cdr conform) alist))
    (dolist (pair alist)
      (rplacd pair (simpHasPred (cdr pair) domform)))
    (dolist (pair alist (nreverse result))
      (setq pred (cdr pair))
      (when pred (setq result (cons pair result))))))

---

defun kcaPage

(defun kcaPage (htPage junk)
  (declare (ignore junk))
  (|kcaPage1| htPage "category" " an "
    "ancestor" #'|ancestorsOf| nil))

---

#define  kcdPage

(defun kcdPage (htPage junk)
  (declare (ignore junk))
  (|kcaPage1| htPage "category" " a "
    "descendant" #'|descendantsOf| t))
defun kcdoPage
[kcdoPage p1521]
[domainsOf p ??]

--- defun kcdoPage ---

(defun kcdoPage (htPage junk)
 (declare (ignore junk))
 (kcPage1 htPage "domain" " a 
 "descendant" #'domainsOf nil))

---

defun kcaPage1
[kcaPage1 p1408]
[kDomainName p1528]
[errorPage p ??]
[form2HtString p ??]
[htpSetProperty p1408]
[opOf p ??]
[augmentHasArgs p1523]
[listSort p ??]
[function p ??]
[dbShowCons p1545]

--- defun kcaPage1 ---

(defun kcaPage1 (htPage kind article whichever fn isCatDescendants?)
 (declare (ignore article))
 (let ((lt1 name nargs domname heading conform conname ancestors choice)
   (setq lt1 (htpProperty htPage '|parts|))
   (setq kind (first lt1))
   (setq name (second lt1))
   (setq nargs (third lt1))
   (setq domname (kDomainName htPage kind name nargs))
   (cond ((and (consp domname) (eq (qcar domname) '|error|))
       (errorPage htPage domname))
     (t
      (setq heading
        (if (null domname)
          (htpProperty htPage '|heading|)
          (list "{\sf " (form2HtString domname nil t) "}")))
        (when (and domname (null isCatDescendants?)))
defun kccPage

(htpProperty p1408)
(kDomainName p1528)
(qcar p??)
[errorPage p??]
(form2HtString p??)
(htpSetProperty p1408)
[opOf p??]
[htInitPage p1415]
[htCopyProplist p??]
[augmentHasArgs p1523]
[childrenOf p??]
[reduceAlistForDomain p1519]
[dbShowCons p1545]

-----

(defun kccPage (htPage junk)
(declare (ignore junk))
(let (lt1 kind name nargs domname heading conform conname page children)
(setq lt1 (htpProperty htPage '|parts|))
(setq kind (first lt1))
(setq name (second lt1))
(setq nargs (third lt1))
(setq domname (kDomainName htPage kind name nargs))
(cond
  ((and (consp domname) (eq (qcar domname) '|error|))
   (errorPage htPage domname))
  (t
   (setq heading

})(htpSetProperty htPage '|domname| domname)
(htpSetProperty htPage '|heading| heading))
(setq conform (htpProperty htPage '|conform|))
(setq conname (opOf conform))
(setq ancestors (FUNCALL fn conform domname))
(unless (string= whichever "ancestor")
  (setq ancestors (augmentHasArgs ancestors conform)))
(setq ancestors (listSort #'glesseqp ancestors))
(htpSetProperty htPage '|cAlist| ancestors)
(htpSetProperty htPage '|thing| whichever)
(setq choice '|names|)
(dbShowCons htPage choice))))
43.2. BRANCHES OF CONSTRUCTOR PAGE

(if (null domname)
  (list "{\sf " (format2HtString domname nil t) "}"
        (|htpSetProperty| htPage '|heading| heading))
(when domname
  (|htpSetProperty| htPage '|domname| domname)
  (|htpSetProperty| htPage '|heading| heading))
(setq conform (|htpProperty| htPage '|conform|))
(setq conname (|opOf| conform))
(setq page
  (|htInitPage| (cons "Children of " heading) (|htCopyProplist| htPage)))
(setq children (|augmentHasArgs| (|childrenOf| conform) conform))
(when domname
  (setq children (reduceAlistForDomain children domname conform)))
(|htpSetProperty| htPage '|cAlist| children)
(|htpSetProperty| htPage '|thing| "child")
(|dbShowCons| htPage '|names|))))

——

(defun augmentHasArgs
  [opOf p??]
  [length p??]
  [nreverse0 p??]
  [extractHasArgs p??]
  [getConstructorForm p??]
  
  — defun augmentHasArgs —

(defun |augmentHasArgs| (alist conform)
  (let (conname args n name p result pred)
    (setq conname (|opOf| conform))
    (setq args (ifcdr conform))
    (cond
      (args
        (setq n (|#| args))
        (dolist (item alist (nreverse0 result))
          (setq name (car item))
          (setq p (cdr item))
          (setq pred
            (if (consp (|extractHasArgs| p))
              p
              (|quickAnd| p
                (cons '|hasArgs|
                  (take n (ifcdr (|getConstructorForm| (|opOf| name)))))))))
        (setq result (cons (cons name pred) result)))
      (t alist))))
defun kcdePage

(htProperty page)
(declare (ignore junk))
(let (lt1 kind name args conname constring conform pakname domlist cAlist)
  (setq lt1 (htpProperty page 'parts))
  (setq kind (first lt1))
  (setq name (second lt1))
  (setq args (sixth lt1))
  (setq conname (intern name))
  (setq constring (concat name args))
  (setq conform
    (if (nequal kind "default package")
      (ncParseFromString constring)
      (cons (intern name) (cdr (ncParseFromString (concat #\d args)))))))
  (setq pakname (opOf conform))
  (setq domlist (getDependentsOfConstructor pakname))
  (setq cAlist
    (loop for x in domList collect (cons (getConstructorForm x) t)))
  (htpSetProperty page 'cAlist cAlist)
  (htpSetProperty page 'thing "dependent")
  (dbShowCons page 'names)))

defun getDependentsOfConstructor

(readLibPathFast)
(pathname page)
— defun getDependentsOfConstructor —

(defun getDependentsOfConstructor (con)
  (let (stream val)
    (setq stream
      (readLibPathFast (pathname (list 'dependents 'database 'a))))
    (setq val (rread con stream nil))
    (rshut stream)
    val))

— defun kcuPage —

(defun kcuPage (htPage junk)
  (declare (ignore junk))
  (let (lt1 kind name args constring conform pakname domlist cAlist)
    (setq lt1 (htpProperty htPage 'parts))
    (setq kind (first lt1))
    (setq name (second lt1))
    (setq args (sixth lt1))
    (setq constring (concat name args))
    (setq conform
      (if (nequal kind "default package")
        (incParseFromString constring)
        (cons (intern name)
          (cdr (incParseFromString (concat #\d args))))))
    (setq pakname
      (if (string= kind "category")
        (intern (concat name #\&))
        (opOf conform)))
(setq domlist (|getUsersOfConstructor| pakname))
(setq cAlist
  (loop for x in domlist collect (cons (|getConstructorForm| x) t)))
(|htpSetProperty| htPage '|cAlist| cAlist)
(|htpSetProperty| htPage '|thing| "user")
(|dbShowCons| htPage '|names|)))

---

(defun getUsersOfConstructor

[readLibPathFast p??]
[pathname p1192]
[rread p847]
[rshut p??]

—- defun getUsersOfConstructor —-

(defun |getUsersOfConstructor| (con)
(\et (stream val)
  (setq stream (|readLibPathFast| (|pathname| (list '|users| 'database '|a|))))
  (setq val (|rread| con stream nil))
  (rshut stream)
  val))

---

(defun kcnPage

[kDomainName p1528]
[qcar p??]
[errorPage p??]
[htpProperty p1408]
[form2HtmlString p??]
[htpSetProperty p1408]
[concat p1197]
[pname p1195]
[opOf p??]
[getImports p??]
[sublis p??]
[dbShowCons p1545]

—- defun kcnPage —-
43.2. BRANCHES OF CONSTRUCTOR PAGE
(defun |kcnPage| (htPage junk)
(declare (ignore junk))
(let (lt1 kind name nargs domname heading conform pakname domlist cAlist
conname)
(setq lt1 (|htpProperty| htPage ’|parts|))
(setq kind (first lt1))
(setq name (second lt1))
(setq conname (intern name))
(setq nargs (third lt1))
(setq domname (|kDomainName| htPage kind name nargs))
(cond
((and (consp domname) (eq (qcar domname) ’|error|))
(|errorPage| htPage domname))
(t
(setq heading
(if (null domname)
(|htpProperty| htPage ’|heading|)
(list "{\\sf " (|form2HtString| domname nil t) "}")))
(if domname
(|htpSetProperty| htPage ’|domname| domname)
(|htpSetProperty| htPage ’|heading| heading))
(setq conform (|htpProperty| htPage ’|conform|))
(setq pakname
(if (string= kind "category")
(intern (concat (pname conname) #\&))
(|opOf| conform)))
(setq domlist (|getImports| pakname))
(when domname
(setq domlist
(sublislis (cons domname (cdr domname))
(cons ’$ (cdr conform)) domlist)))
(setq cAlist (loop for x in domList collect (cons x t)))
(|htpSetProperty| htPage ’|cAlist| cAlist)
(|htpSetProperty| htPage ’|thing| "benefactor")
(|dbShowCons| htPage ’|names|)))))

———-

defun koPageInputAreaUnchanged?
[htpLabelInputString p1409]
[concat p1197]
[htpProperty p1408]
— defun koPageInputAreaUnchanged? —
(defun |koPageInputAreaUnchanged?| (htPage nargs)

1527


(equal
  (loop for i from 1 to nargs
    collect
    (|htpLabelInputString| htPage (intern (concat "\*" (princ-to-string i)))))
  (|httpProperty| htPage '|inputAreaList|)))

---

defun kDomainName

[httpSetProperty p1408]
[httpLabelInputString p1409]
[getdatabase p1156]
[kArgumentCheck p1529]
[concat p1197]
[unabbrev p??]
[mkConform p1532]
[kisIsValidType p1530]
[dbMkEvalable p1529]
[spad-reader p??]
[$PatternVariableList p??]

— defun kDomainName —

(defun |kDomainName| (htPage kind name nargs)
  (let (inputAreaList conname args n argTailPart argString typeForm
        evaluatedTypeForm)
    (|httpSetProperty| htPage '|domname| nil)
    (setq inputAreaList
      (loop for i from 1 to nargs for var in |$PatternVariableList|
        collect (|htpLabelInputString| htPage var)))
    (|httpSetProperty| htPage '|inputAreaList| inputAreaList)
    (setq conname (intern name))
    (setq args
      (loop for x in inputAreaList
        for domain? in (cdr (getdatabase conname 'cosig))
        collect (or (|kArgumentCheck| domain? x) nil)))
    (when (some #'identity (loop for x in args collect (null x)))
      (cond
        (> (setq n (apply #'+ (loop for x in args collect (if x 1 0)))) 0)
        (list '|error| nil "\centerline{You gave values for only \{\em \n \} of the \{\em (#| args) \}}"
          "\centerline{parameters of \{\sf \name \}}\vspace{1}\centerline{Please enter either \{\em all\} or "
          "\{\em none\} of the type parameters\"}
          nil)
      (t
(setq argString
  (cond
    ((null args) "()")
    (t
     (setq argTailPart
       (apply '#concat
         (loop for x in (ifcdr args) collect (concat (cons ""," x))))))
     (apply '#concat (list "(" (car args) argTailPart ")")))
     (setq typeForm
       (or (catch 'spad_reader (|unabbrev| (|mkConform| kind name argString)))
         (list '|error| '|invalidType| (concat name argString))))
     (if (null (setq evaluatedTypeForm (|kisValidType| typeForm)))
       (list '|error| '|invalidType| (concat name argString))
       (|dbMkEvalable| evaluatedTypeForm)))))))

———

defun kArgumentCheck

[conSpecialString? p1502]
[opOf p??]
[form2String p??]

—— defun kArgumentCheck ——

(defun |kArgumentCheck| (domain? s)
  (let (form)
    (cond
      ((string= s "") nil)
      ((and domain? (setq form (|conSpecialString?| s)))
       (if (null (ifcdr form))
         (list (princ-to-string (|opOf| form)))
         (|form2String| form)))
      (t (list s)))))

———

defun dbMkEvalable

[getdatabase p1156]
[inkEvalable p1075]

—— defun dbMkEvalable ——

(defun |dbMkEvalable| (form)
(let (op kind)
  (setq op (car form))
  (setq kind (getdatabase op 'constructorkind))
  (if (eq kind '|category|)
      form
      (|mkEvalable| form)))

defun topLevelInterpEval

[processInteractive p274]
[$ProcessInteractiveValue p276]
[$noEvalTypeMsg p1082]

— defun topLevelInterpEval —

(defun |topLevelInterpEval| (x)
  (let (|$ProcessInteractiveValue| |$noEvalTypeMsg| it1)
    (declare (special |$ProcessInteractiveValue| |$noEvalTypeMsg|))
    (setq |$ProcessInteractiveValue| t)
    (setq |$noEvalTypeMsg| t)
    (|processInteractive| x nil)))

defun kisValidType

[processInteractive p274]
[member p1198]
[kCheckArgumentNumbers p1531]
[$ProcessInteractiveValue p276]
[$noEvalTypeMsg p1082]
[spad-reader p??]

— defun kisValidType —

(defun |kisValidType| (typeForm)
  (let (|$ProcessInteractiveValue| |$noEvalTypeMsg| it1)
    (declare (special |$ProcessInteractiveValue| |$noEvalTypeMsg|))
    (setq |$ProcessInteractiveValue| t)
    (setq |$noEvalTypeMsg| t)
    (setq it1 (catch 'spad_reader (|processInteractive| typeForm nil)))
    (when (and (consp it1) (consp (qcar it1)))
(member (caar it1) `(domain |SubDomain|)))
(and (kCheckArgumentNumbers (qcdr it1)) (qcdr it1)))

---

defun kCheckArgumentNumbers

[getdatabase p1156]
[kCheckArgumentNumber p??]

---
defun kCheckArgumentNumbers ---

(defun |kCheckArgumentNumbers| (tt)
  (let (conname args cosig)
    (setq conname (car tt))
    (setq args (cdr tt))
    (setq cosig (ifcdr (getdatabase conname 'cosig))
      (every #'identity
        (loop for domain? in cosig for x in args
          collect (if domain? (|kCheckArgumentNumbers| x) t))))))

---

defun parseNoMacroFromString

[next p262]
[function p??]
[ncloopParse p262]
[lineoftoks p335]
[incString p263]
[StreamNull p555]
[pf2Sex p529]

---
defun parseNoMacroFromString ---

(defun |parseNoMacroFromString| (s)
  (setq s
    (|next| #'|ncloopParse|
      (|next| #'|lineoftoks|
        (|incString| s)))))
(if (|StreamNull| s)
   nil
   (|pf2Sex| (cadar s))))

---
defun mkConform

(nequal p??)  
(parseNoMacroFromString p1531)  
(sayBrightlyNT p??)  
(pp p??)  
(systemError p??)  
(ncParseFromString p1240)  
(concat p1197)

— defun mkConform —

(defun |mkConform| (kind name argString)
  (let ((form parse))
    (cond
      ((nequal kind "default package")
        (setq form (concat name argString))
        (setq parse (|parseNoMacroFromString| form))
        (cond
          (null parse)
            (|sayBrightlyNT| "Won't parse: ")
            (|pp| form)
            (|systemError| "Keywords in argument list?"))
          ((atom parse) (cons parse nil))
          (t parse))
        (t
          (cons (intern name) (cdr (|ncParseFromString| (concat #\d argString))))))))

43.3 Operation Page for a Domain Form from Scratch

defun conOpPage

(dbCompositeWithMap p1534)  
(htpProperty p1408)  
(conOpPage1 p1533)  
(dbExtractUnderlyingDomain p1535)

— defun conOpPage —

(defun |conOpPage| (htPage conform)
  (declare (ignore conform))
  (let (updown domname)
    (setq updown (|dbCompositeWithMap| htPage)))
43.3. **OPERATION PAGE FOR A DOMAIN FORM FROM SCRATCH**

```
(defun conOpPage1 (&rest arg)
  (let (bindingsAlist conname domname line parts name sig args.isFile kind
        constring capitalKind signature sourceFileName emString heading page
        selectedOperation options conform)
    (declare (special "$Primitives"))
    (setq conform (car arg))
    (setq options (cdr arg))
    (setq bindingsAlist (ifcar options))
    (setq conname (opOf conform))
    (cond
      ((member conname "$Primitives") (dbSpecialOperations conname))
      (t
       (setq domname (unless (atom conform)
                        (|conOpPage1| domname nil))))))
```

---

**defun conOpPage1**

```
(defun conOpPage1 ()
  (ifcar p [[]]
    [opOf p []]
    [dbSpecialOperations p 1556]
    [conPageFastPath p 1504]
    [dbXParts p []]
    [concat p 1197]
    [mkConform p 1532]
    [captialize p []]
    [ncParseFromString p 1240]
    [dbSourceFile p []]
    [isExposedConstructor p 1026]
    [htInitPage p 1415]
    [htpSetProperty p 1408]
    [lassoc p []]
    [ifcdr p []]
    [koPage p 1535]
    [$Primitives p []]

  — defun conOpPage1 —
```

```
(defun conOpPage1 () (krest arg)
  (let (bindingsAlist conname domname line parts name sig args.isFile kind
        constring capitalKind signature sourceFileName emString heading page
        selectedOperation options conform)
    (declare (special "$Primitives"))
    (setq conform (car arg))
    (setq options (cdr arg))
    (setq bindingsAlist (ifcar options))
    (setq conname (opOf conform))
    (cond
      ((member conname "$Primitives") (dbSpecialOperations conname))
      (t
       (setq domname (unless (atom conform)
                        (|conOpPage1| domname nil))))))
```
(setq line (|conPageFastPath| conname))
(setq parts ([dbXParts| line 7 1]))
(setq kind (first parts))
(setq name (second parts))
(setq sig (fifth parts))
(setq args (sixth parts))
(setq isFile (null kind))
(setq kind (or kind "package"))
(rplaca parts kind)
(setq constring (concat name args))
(setq conform ([mkConform| kind name args]))
(setq capitalKind ([capitalize| kind]))
(setq signature ([ncParseFromString| sig]))
(setq sourceFileName ([dbSourceFile| (intern name)]))
(setq emString (list "{{\sf " constring "])")
(setq heading (cons capitalKind (cons " " emString)))
(unless ([isExposedConstructor| conname])
  (setq heading (cons "Unexposed " heading)))
(setq page ([htInitPage| heading nil]))
([ht SetProperty| page ]|isFile| t)
([ht SetProperty| page ]|fromConOpPage1| t)
([ht SetProperty| page ]|parts| parts)
([ht SetProperty| page ]|heading| heading)
([ht SetProperty| page ]|kind| kind)
([ht SetProperty| page ]|domname| domname)
([ht SetProperty| page ]|conform| conform)
([ht SetProperty| page ]|signature| signature)
(when
  (setq selectedOperation (lassoc '|selectedOperation| (ifcdr options)))
  ([ht SetProperty| page ]|selectedOperation| selectedOperation))
(loop for item in bindingsAlist
  collect ([ht SetProperty| page |car item| (cdr item)])
([koPage| page "operation"))))

---

defun dbCompositeWithMap

[ht SetProperty p1408]
[db ExtractUnderlyingDomain p1535]

— defun dbCompositeWithMap —

(defun |dbCompositeWithMap| (htPage)
  (let (domain opAlist)
    (cond
     (([ht SetProperty| htPage ]|updomain|) "UP")


defun dbExtractUnderlyingDomain

[isValidType p??]

— defun dbExtractUnderlyingDomain —

(defun |dbExtractUnderlyingDomain| (domain)
  (some #'identity
    (loop for x in (ifcdr domain) when (isValidType x) collect x)))

43.4 Operation Page from Main Page

defun koPage

[htpProperty p1408]
[concat p1197]
[koPageInputAreaUnchanged? p1527]
[kDomainName p1528]
[errorPage p??]
[form2HtString p??]
[capitalize p??]
[htpSetProperty p1408]
[koPageAux p1536]

— defun koPage —

(defun |koPage| (htPage which)
  (let ((lt1 kind name nargs args constring conname u IT1 domname headingString heading)

(t
  (setq domain (htProperty htPage 'domname))
  (cond
    ((null domain) nil)
    (t
      (setq opAlist (htProperty htPage 'opAlist))
      (when
        (dbExtractUnderlyingDomain (htProperty htPage 'domname))
        "DOWN"))))))

———
(setq lt1 (htpProperty htPage 'parts))
(setq kind (first lt1))
(setq name (second lt1))
(setq nargs (third lt1))
(setq args (sixth lt1))
(setq constring (concat name args))
(setq conname (intern name))
(setq IT1 (setq u (htpProperty htPage 'domname)))
(setq domname
  (cond
    ((and (consp IT1) (equal (qcar IT1) conname)
      (or (eq (htpProperty htPage 'fromConOpPage1) t)
        (koPageInputAreaUnchanged? htPage nargs)))
     u)
    (t (kDomainName htPage kind name nargs))))
  (cond
    ((and (consp domname) (eq (qcar domname) 'error))
     (errorPage htPage domname))
    (t
     (htpSetProperty htPage 'domname domname)
     (setq headingString (if domname (form2HtString domname nil t) constring))
     (setq heading (list (capitalize kind) " {\sf " headingString "} "))
     (htpSetProperty htPage 'which which)
     (htpSetProperty htPage 'heading heading)
     (koPageAux htPage which domname heading))))

——

(defun koPageFromKKPage
  [koPageAux p1536]
  [htpProperty p1408]

  — defun koPageFromKKPage —

  (defun koPageFromKKPage (htPage ao)
    (koPageAux htPage ao (htpProperty htPage 'domname)
      (htpProperty htPage 'heading))))

——

(defun koPageAux
  [htpSetProperty p1408]
  [koAttrs p??]
defun koPageAux (htPage which domname heading)
  (let (conform selectedOperation opAlist)
    (htpSetProperty htPage '|which| which)
    (setq domname (htpProperty htPage '|domname|))
    (setq conform (htpProperty htPage '|conform|))
    (setq heading (htpProperty htPage '|heading|))
    (setq opAlist
      (cond
        (setq selectedOperation (htpProperty htPage '|selectedOperation|))
        (setq opAlist
          (list (or (assoc selectedOperation opAlist) (systemError)))))
    (dbShowOperationsFromConform htPage which opAlist)))

defun koPageAux1 (htPage opAlist)
  (let (which)
    (setq which (htpProperty htPage '|which|))
    (dbShowOperationsFromConform htPage which opAlist)))

defun koaPageFilterByName
(defun koaPageFilterByName (htPage functionToCall)
  (let (filter which opAlist)
    (cond
     ((string= (hntpLabelInputString htPage 'filter) "")
      (koaPageFilterByCategory htPage functionToCall))
     (t
      (setq filter (pmTransFilter (hdbGetInputString htPage)))
      (setq which (hntpProperty htPage 'which))
      (setq opAlist
        (loop for x in (hntpProperty htPage 'opAlist)
          when (superMatch? filter (downcase (princ-to-string (car x))))
          collect x))
      (hntpSetProperty htPage 'opAlist opAlist)
      (funcall functionToCall htPage nil))))

43.5 Get Constructor Documentation

defun dbConstructorDoc,hn

(defun dbConstructorDoc,hn\ (sig)
  (declare (special |$sig| |$args|))
  (and (equal (|#| |$sig|) (|#| sig))
    (equal $sig (sublisls |$args| |$FormalMapVariableList| sig))))
defun dbConstructorDoc, gn

[dbConstructorDoc, hn p1538]
[$op p??]

— defun dbConstructorDoc, gn —

(defun |dbConstructorDoc, gn| (arg)
(let (op alist)
(declare (special |$op|))
(setq op (car arg))
(setq alist (cdr arg))
(and |$op|
(some #'identity
(loop for item in alist when (|dbConstructorDoc, hn| (car item))
collect (or (cdr item) ""))))))

---

defun dbConstructorDoc

[dbConstructorDoc, fn p??]
[$sig p??]
[$op p??]

— defun dbConstructorDoc —

(defun |dbConstructorDoc| (conform |$op| |$sig|)
(declare (special |$op| |$sig|))
(|dbConstructorDoc, fn| conform))

---

defun dbDocTable

[lget p1194]
[make-hashtable p??]
[originsInOrder p1540]
[dbAddDocTable p1540]
[$docTable p??]
[$docTableHash p??]

— defun dbDocTable —
(defun dbDocTable (conform)
  (let ([$docTable| table])
    (declare (special $docTable| $docTableHash|))
    (cond
      ((setq table (hget $docTableHash| conform))
        table)
      (t
        (setq $docTable| (make-hashtable 'id))
        (loop for x in (originsInOrder conform) do (dbAddDocTable| x))
        (dbAddDocTable| conform)
        (hput $docTableHash| conform $docTable|)
        $docTable|))))

---

defun originsInOrder

[getdatabase p1156]
[assocleft p??]
[ancestorsOf p??]
[parentsOf p??]
[originsInOrder p1540]
[insert p??]

— defun originsInOrder —

(defun originsInOrder (conform)
  (let (con argl acc)
    (setq con (car conform))
    (setq argl (cdr conform))
    (cond
      ((eq (getdatabase con 'constructorkind) 'category)
        (assocleft (ancestorsOf conform nil)))
      (t
        (setq acc (assocleft (parentsOf conform)))
        (loop for x in acc do
          (loop for y in (originsInOrder x) do
            (setq acc (insert y acc)))
          acc)))))

---

defun dbAddDocTable

[opOf p??]
[getConstructorForm p??]
43.5. GET CONSTRUCTOR DOCUMENTATION

(defun dbAddDocTable (conform)
  (let (conname storedArgs op alist op1 sig doc tmp)
    (declare (special $docTable))
    (setq conname (opOf conform))
    (setq storedArgs (cdr (getConstructorForm conname)))
    (setq tmp (sublislis (cons '$ (cdr conform)) (cons '% storedArgs)
                       (getdatabase (opOf conform) 'documentation)))
    (loop for item in tmp do
      (setq op (car item))
      (setq alist (cdr item))
      (setq op1
        (cond
         ((eq op '(|Zero|)) 0)
         ((eq op '(|One|)) 1)
         (t op)))
      (loop for item1 in alist do
        (setq sig (first item1))
        (setq doc (second item1))
        (hput $docTable op1 (cons (cons conform alist)
                                  (hget $docTable op1)))))
    )

(defun dbGetDocTable,hn (arg)
  (let (sig doc alteredSig pred)
    (declare (special $which $conform $sig $FormalMapVariableList))
    )

(setq sig (car arg))
(setq doc (cdr arg))
(if (string= |$which| "attribute")
  (and (consp sig) (eq (qcar sig) '|attribute|) (equal (qcdr sig) |$sig|) doc)
  (progn
    (setq pred
      (and
        (eql (|#| |$sig|) (|#| sig))
        (setq alteredSig
          (sublislis (ifcdr |$conform|) |$FormalMapVariableList| sig))
        (equal alteredSig |$sig|)))
    (when (and pred doc
       (and (consp pred) (eq (qcar pred) '|constant|)) (qcdr pred) pred)
       ,"")))
)

defun dbGetDocTable,gn

[|lastatom p||]
[|dbGetDocTable,hn p1541|
[|$conform p||]

— defun dbGetDocTable,gn —

(defun |dbGetDocTable,gn| (u)
(let (code p comments)
  (declare (special |$conform|))
  (setq |$conform| (car u))
  (when (atom |$conform|) (setq |$conform| (list |$conform|)))
  (setq code (lastatom u))
  (setq comments
    (some #'identity
      (loop for entry in (cdr u)
        when (setq p ([|dbGetDocTable,hn| entry])
          collect p)))
    (when comments (cons |$conform| (cons (car comments) code))))))

—

defun dbGetDocTable

[string2Integer p||]
[|dbConstructorDoc p1539|]
43.5. GET CONSTRUCTOR DOCUMENTATION

--- defun dbGetDocTable ---

(defun dbGetDocTable (op sig docTable which aux)
  (declare (special sig which))
  (let (doc origin s)
    (declare (special conform op))
    (when (and (null (integerp op)) (digitp (elt (setq s (princ-to-string op)) 0)))
      (setq op (string2Integer s)))
    (cond
      ((and (consp aux) (consp (qcar aux)))
       (setq doc (dbConstructorDoc (car aux) op sig))
       (setq origin (if (qcdr aux) (cons 'ifp aux) (car aux)))
       (cons origin doc))
      (t
       (some #'identity
         (loop for x in (hget docTable op)
               collect (dbGetDocTable,gn x))))))

--- defun kTestPred ---

defun kTestPred

(defun kTestPred (n)
  (declare (special predvec domain))
  (cond
    ((eql n 0) t)
    (domain (testBitVector predvec n))
    (t (simpHasPred (elt predvec (1- n))))))
defun dbAddChainDomain

(defun |dbAddChainDomain| (conform)
  (let (name args template form)
    (declare (special |$infovec|))
    (setq name (car conform))
    (setq args (cdr conform))
    (setq |$infovec| (|dbInfovec| name))
    (when |$infovec|
      (setq template (elt |$infovec| 0))
      (when (setq form (elt template 5))
        (|dbSubConform| args (|kFormatSlotDomain| (|devaluate| form)))))))

defun dbSubConform

(defun |dbSubConform| (args u)
  (let (n y)
    (declare (special |$FormalMapVariableList|))
    (cond
      ((atom u)
        (if (>= (setq n (|position| u |$FormalMapVariableList|)) 0)
            (elt args n)
            u))
      ((and (consp u) (eq (car u) '|local|) (consp (cdr u)) (eq (cddr u) nil))
        (setq y (cadr u))
        (|dbSubConform| args y))
      (t
        (loop for x in u collect (|dbSubConform| args x))))))
43.6. CONSTRUCTOR PAGE MENU

defun dbAddChain

(defun |dbAddChain| (conform)
  (let (u)
    (when (setq u (|dbAddChainDomain| conform))
      (unless (atom u)
        (cons (cons u t) (|dbAddChain| u)))))

—— defun dbAddChain ——

43.6 Constructor Page Menu

defun dbShowCons

(defun |dbShowCons| (&rest args)
  (let (cAlist filter abbrev? conname subject u options key htPage)
    (declare (special |$exposedOnlyIfTrue|))
    (setq htPage (first args))
    (setq key (second args))
    (setq options (cddr args))
    (setq cAlist (|htpProperty| htPage '|cAlist|)))

—— defun dbShowCons ——
(cond
  ((eq key '|filter|)
   (setq filter
     (|pmTransFilter| (or (ifcar options) (|dbGetInputString| htPage)))))
  (cond
    ((and (consp filter) (eq (car filter) '|error|))
     (|bcErrorPage| filter))
    (t
     (setq abbrev? (eq (|htpProperty| htPage '|exclusion|) '|abbrs|))
      (setq u
        (loop for x in cAlist
              when (progn
                (setq conname (caar x))
                (setq subject (if abbrev? (|constructor?| conname) conname))
                (|superMatch?| filter (downcase (princ-to-string subject))))
              collect x))
     (cond
       ((null u)
        (|emptySearchPage| "constructor" filter))
       (t
        (setq htPage (|htInitPageNoScroll| (|htCopyProplist| htPage))
          (|htpSetProperty| htPage '|cAlist| u)
          (|dbShowCons| htPage (|htpProperty| htPage '|exclusion|)))))))
  (t
   (when (member key '(|exposureOn| |exposureOff|))
    (setq|$exposedOnlyIfTrue| (eq key '|exposureOn|))
     (setq key (|htpProperty| htPage '|exclusion|))
     (|dbShowCons1| htPage cAlist key))))

----

defun conPageChoose

[getConstructorForm p??]
[dbShowCons1 p1547]

--- defun conPageChoose ---

(defun |conPageChoose| (conname)
  (let (cAlist)
    (setq cAlist (list (cons (|getConstructorForm| conname) t)))
    (|dbShowCons1| nil cAlist '|names|)))

---
defun dbShowCons1

[remdup p??]
[isExposedConstructor p1026]
[opOf p??]
[conPage p1503]
[htpProperty p1408]
[union p??]
[dbConstructorKind p??]
[htCopyProplist p??]
[htInitPageNoScroll p??]
[dbConsHeading p1553]
[htSayStandard p??]
[htpsetProperty p1408]
[bcNameConTable p??]
[bcAbbTable p??]
[getCDTEntry p??]
[getdatabase p1156]
[bcUnixTable p1554]
[listSort p??]
[function p??]
[qlesseqp p??]
[dbShowConsDoc p1549]
[isExposedConstructor p1026]
[dbShowConditions p1552]
[bcConTable p??]
[assocleft p??]
[dbShowConsKinds p??]
[dbConsExposureMessage p1549]
[dbPresentCons p??]
[htShowPageNoScroll p149]
[$conformsAreDomains p??]
[$exposedOnlyIfTrue p??]

— defun dbShowCons1 —

(defun |dbShowCons1| (htPage cAlist key)
  (let ((|$conformsAreDomains| conlist kinds kind proplist page u flist result)
    (declare (special |$conformsAreDomains| |$exposedOnlyIfTrue|))
    (setq conlist
      (remdup
        (dolist x cAlist result)
        (push
          (if |$exposedOnlyIfTrue|
            (if (isExposedConstructor |opOf| (car x))
              (car x)
              result))))))
(cond
  ((and (consp conlist) (eq (qcdr conlist) nil))
   (conPage)
   (if (and htPage (|htpProperty| htPage '|domname|))
     (car conlist)
     (|opOf| (car conlist))))
  (t
   (setq conlist (loop for x in conlist collect (|opOf| x)))
   (setq kinds
     (apply #'|union|
       (loop for x in conlist collect (|dbConstructorKind| x))))
   (setq kind
     (if (and (consp kinds) (eq (qcdr kinds) nil))
       (qcar kinds)
       '|constructor|)
     (setq proplist (when htPage (|htCopyProplist| htPage)))
   (setq page
     (|htInitPageNoScroll| proplist
       (|dbConsHeading| htPage conlist key kind))
   (if (setq u (|htpProperty| page '|specialMessage|))
     (apply (car u) (cdr u)))
   (|htSayStandard| "\beginscroll ")
   (|htpSetProperty| page '|cAlist| cAlist)
   (setq $conformsAreDomains| (|htpProperty| page '|domname|))
   (cond
     ((eq (qcar kind) '|names|) (|bcNameConTable| conlist))
     ((eq (qcar kind) '|abbrs|)
      (|bcAbbTable|
       (loop for con in conlist collect (|getCDTEntry| con t))))
     ((eq (qcar kind) '|files|)
      (setq flist
        (loop for con in conlist collect (getdatabase con 'sourcefile)))
      (|bcUnixTable|
        (|listSort| #'glesseqp (remdup flist)))
      (setq documentation| (|dbShowConsDoc| page conlist))
     (t
      (when $exposedOnlyIfTrue|
       (setq cAlist
         (loop for x in cAlist
d          when (|isExposedConstructor| (qcar x))
d          collect x))
      (cond
       ((eq (qcar kind) '|conditions|) (|dbShowConditions| page cAlist kind))
       ((eq (qcar kind) '|parameters|)
        (|bcConTable| (remdup (assocleft cAlist))))
       ((eq (qcar kind) '|kinds|) (|dbShowConsKinds| cAlist))
      )
      (|dbConsExposureMessage|)
      (|htSayStandard| "\endscroll ")
      (|dbPresentCons| page kind key)
      (|htShowPageNoScroll|)))))
defun dbConsExposureMessage

[htSay p??]
[$atLeastOneUnexposed p??]

— defun dbConsExposureMessage —

(defun |dbConsExposureMessage| ()
  (declare (special |$atLeastOneUnexposed|))
  (when |$atLeastOneUnexposed|
    (|htSay| "\newline{}---------\newline{}\ms* unexposed")))

— defun dbShowConsKindsFilter —

(defun |dbShowConsKindsFilter| (htPage args)
  (|htpSetProperty| htPage '|cAlist| (second args))
  (|dbShowCons| htPage (|htpProperty| htPage '|exclusion|)))

— defun dbShowConsDoc —

(defun |dbShowConsDoc|)

[systemError p??]
[dbShowConsDoc1 p1550]
[getConstructorForm p??]
[opOf p??]
[htpProperty p1408]
[remdup p??]
(defun |dbShowConsDoc| (htPage conlist)
  (labels (fn (cAlist x)
    (let ((index 0))
      (loop while (not (equal (caaar cAlist) x))
        do (setq index (1+ index))
        (setq cAlist (cdr cAlist))
        (unless cAlist (|systemError|))
        index)))
  (let (cAlist)
    (cond
      ((null (cdr conlist))
       (|dbShowConsDoc1| htPage (|getConstructorForm| (|opOf| (car conlist))) nil))
      (t
       (setq cAlist (|htpProperty| htPage '|cAlist|))
       (loop for x in (remdup conlist) do
         (|dbShowConsDoc1| htPage (|getConstructorForm| x) (fn cAlist x)))))))
  
  —— defun |dbShowConsDoc1| ——

(defun |dbShowConsDoc1| (htPage conform indexOrNil)
  (let (conargs conname lt1 exposeFlag doc signature sig)
    (declare (special |$TriangleVariableList| |$Primitives|)
      [member p1198]
      [htpProperty p1408]
      [getl p1200]
      [displayDomainOp p??]
      [isExposedConstructor p1026]
      [getConstructorDocumentation p1551]
      [getConstructorSignature p??]
      [getdatabase p1156]
      [sublis p??]
      [sublisFormal p??]
      [displayDomainOp p??]
      [$TriangleVariableList p??]
      [$Primitives p??]
    (let (conname (car conform) lt1 (cdr conform) exposeFlag doc signature sig)
      (declare (special |$TriangleVariableList| |$Primitives|)
        [member p1198]
        [htpProperty p1408]
        [getl p1200]
        [displayDomainOp p??]
        [isExposedConstructor p1026]
        [getConstructorDocumentation p1551]
        [getConstructorSignature p??]
        [getdatabase p1156]
        [sublis p??]
        [sublisFormal p??]
        [displayDomainOp p??]
        [$TriangleVariableList p??]
        [$Primitives p??]
      (let (conargs conslist indexOrNil)
        (cond
          ((null (cdr conslist))
           (|dbShowConsDoc1| htPage
            (|getConstructorForm| (|opOf| (car conslist))) nil))
          (t
           (setq conslist (|htpProperty| htPage '|conslist|))
           (loop for x in (remdup conslist) do
             (|dbShowConsDoc1| htPage
              (|getConstructorForm| x) (fn conslist x)))))),
      (setq conform (list consname conargs))
      (setq docsignature (list doc signature))
      (setq exposelongflag (list exposeFlag))
      (setq index (list indexOrNil))
      (setq index (if index (car index) (cadr conform))))))

  
  —— defun |dbShowConsDoc1| ——
```lisp
((member conname '$Primitives))
(setq conname (htpProperty htPage 'conname))
(setq lt1 (get1 conname 'documentation))
(cond ((eq (caar lt1) 'constructor) (caar lt1)))
(cond ((eq (caadar lt1) 'nil) (caadar lt1)))
(setq doc (car (cdadar lt1)))
(setq sig `'((category domain) (|SetCategory|) (|SetCategory|))
(displayDomainOp htPage "constructor"
    conform conname sig t doc indexOrNil 'dbSelectCon nil nil))
(t
(setq exposeFlag (|isExposedConstructor| conname))
(setq doc (list (|getConstructorDocumentation| conname)))
(setq signature (|getConstructorSignature| conname))
(setq sig
  (if (eq (getdatabase conname 'constructorkind) 'category)
    (sublislis conargs '$TriangleVariableList signature)
    (sublisFormal conargs signature))
(displayDomainOp htPage "constructor" conform conname sig t doc
    indexOrNil 'dbSelectCon (null exposeFlag) nil)))

---

defun getConstructorDocumentation

[lassoc p??]
[getdatabase p1156]
[qcar p??]
[qcaar p??]
[qcdar p??]
[qcadar p??]

— defun getConstructorDocumentation —

(defun |getConstructorDocumentation| (conname)
  (let (IT1)
    (setq IT1 (lassoc 'constructor (getdatabase conname 'documentation)))
    (or
      (and (consp IT1) (consp (qcar IT1)) (null (qcaar IT1)) (consp (qcdar IT1))
       (qcadar IT1)
      "")))

---

defun dbSelectCon

(defun dbSelectCon (htPage which index)
  (declare (ignore which))
  (conPage (opOf (car (elt (htpProperty htPage 'cAlist) index)))))

defun dbShowConditions

(defun dbShowConditions (htPage cAlist kind)
  (let (conform conname article whichever lt1 consNoPred consPred singular plural)
    (setq conform (htpProperty htPage 'conform))
    (setq conname (opOf conform))
    (setq article (htpProperty htPage 'article))
    (setq whichever (htpProperty htPage 'whichever))
    (setq lt1 (splitConTable cAlist))
    (setq consNoPred (car lt1))
    (setq consPred (cadr lt1))
    (setq singular (list kind "is")
    (setq plural (list (pluralize (princ-to-string kind)) "are"))
    (dbSayItems (list consNoPred singular plural "unconditional")
    (bcConPredTable consNoPred conname)
    (htSayHrule)
    (dbSayItems (list consPred singular plural "conditional")
    (bcConPredTable consPred conname)))

— defun dbSelectCon —

— defun dbShowConditions —
defun dbConsHeading

(defun |dbConsHeading| (htPage conlist view kind)
  (let (thing place count rank modifier exposureWord firstWord prefix
        placepart connective heading)
    (declare (special |$exposedOnlyIfTrue|))
    (setq thing (or (and htPage (|htpProperty| htPage '|thing|)) "constructor"))
    (setq place
      (when htPage
        (or (|htpProperty| htPage '|domname|) (|htpProperty| htPage '|conform|))))
    (setq count (|#| (remdup conlist)))
    (cond
      ((string= thing "benefactor")
       (list (princ-to-string count) " Constructors Used by "
         (|form2HtString| place nil t)))
      (t
        (setq modifier
          (cond
            ((string= thing "argument")
             (setq rank (and htPage (|htpProperty| htPage '|rank|)))
             (list " Possible " rank " ")
             ((eq kind '|constructor|)
              (list " "))
             (t
              (cons " " (|capitalize| (princ-to-string kind)) " ")))))
        (setq exposureWord (when |$exposedOnlyIfTrue| '(" Exposed "))
        (setq prefix
          (cond
            ((eql count 1)
             (cons (princ-to-string count)
               (append modifier (list (|capitalize| thing))))))
            (t
             (setq firstWord (if (eql count 0) "No " (princ-to-string count)))
             (cons firstWord
               (append exposureWord
                (append modifier
                  (list (|capitalize| (|pluralize| thing))))))))))
(setq placepart
  (when place (list " of \textit{" (|form2HtString| place nil t) '})))
(setq heading (append prefix placepart))
(setq connective
  (if (|member| view '(|abbrs| |files| |kinds|)) " as " " with ")
  (cond
    ((and (nequal count 0)
      (|member| view '(|abbrs| |files| |parameters| |conditions|)))
      (setq heading
        (append heading
          (list " viewed" connective "{\textit{ (princ-to-string view) "}"))))
      heading))))
)

defun dbShowConstructorLines

[getConstructorForm p??]
[intern p??]
[dbName p??]
[dbShowCons1 p1547]
[listSort p??]
[function p??]
[glesseqp p??]

— defun dbShowConstructorLines —

(defun |dbShowConstructorLines| (lines)
  (let (cAlist)
    (setq cAlist
      (loop for line in lines
        collect (cons (|getConstructorForm| (|intern| (|dbName| line))) t)))
    (|dbShowCons1| nil (|listSort| #'glesseqp cAlist) '|names|)))

———

defun bcUnixTable

[htSay p??]
[htBeginTable p??]
[namestring p1190]
[findfile p??]
[htMakePage p1416]
[htEndTable p??]
— defun bcUnixTable —

(defun bcUnixTable (u)
  (let (filename)
    (|htSay| "\newline")
    (|htBeginTable|)
    (setq firstTime t)
    (loop for x in u do
      (|htSay| "({"
      (setq filename (namestring ($findfile (princ-to-string x) "SPAD"))))
      (|htMakePage|)
      (list
        (list '|text| "\unixcommand{" (pathname-name x)
          "}\{AXIOM/lib/SPADEDIT " filename "} "))
      (|htSay| "}")
    (|htEndTable|)))

---

Special Code for Union, Mapping, and Record

defun dbSpecialDescription

[getConstructorForm p??]
[form2HtString p??]
[htInitPage p1415]
[htpSetProperty p1408]
[dbShowConsDoc1 p1550]
[htShowPage p1416]
[$conformsAreDomains p??]

— defun dbSpecialDescription —

(defun dbSpecialDescription (conname)
  (let (conform heading page)
    (declare (special $conformsAreDomains))
    (setq conform (|getConstructorForm| conname))
    (setq heading
      (list "Description of Domain {\sf " (|form2HtString| conform) "}")
    (setq page (|htInitPage| heading nil))
    (|htpSetProperty| page '|conname| conname)
    (setq $conformsAreDomains nil)
    (|dbShowConsDoc1| page conform nil)
    (|htShowPage|)))
defun dbSpecialOperations
[htInitPage p1415]
[getConstructorForm p??]
[dbSpecialExpandIfNecessary p1557]
[getl p1200]
[form2HtString p??]
[htpSetProperty p1408]
[dbShowOp1 p??]

---

(defun dbSpecialOperations (conname)
  (let (page conform opAlist fromHeading)
    (setq page (htInitPage nil nil))
    (setq conform (getConstructorForm conname))
    (setq opAlist
      (dbSpecialExpandIfNecessary conform
        (cdr (getl conname '|documentation|))))
    (setq fromHeading (list " from domain \{sf " (form2HtString conform) "}")
    (htpSetProperty page '|fromHeading| fromHeading)
    (htpSetProperty page '|conform| conform)
    (htpSetProperty page '|opAlist| opAlist)
    (htpSetProperty page '|noUsage| t)
    (htpSetProperty page '|condition?| '|no|)
    (dbShowOp1 page opAlist "operation" '|names|)))

---

defun dbSpecialExports
[getConstructorForm p??]
[htInitPage p1415]
[form2HtString p??]
[dbSpecialExpandIfNecessary p1557]
[getl p1200]
[kePageDisplay p1512]
[htShowPage p1416]

---

(defun dbSpecialExports (conname)
  (let (conform page opAlist)
(setq conform (getConstructorForm conname))
(setq page
  (htInitPage (list "Exports of {\sf " (form2HtString conform) "}" nil)))
(setq opAlist
  (dbSpecialExpandIfNecessary conform
    (cdr (getl conname '|documentation|))))
(kePageDisplay page "operation" opAlist)
(htShowPage))

---

(defun dbSpecialExpandIfNecessary
  (conform opAlist)
  (if (and (consp opAlist) (consp (qcar opAlist)) (consp (qcdar opAlist))
    (consp (qcadr opAlist)) (cdr (qcdr (qcadr opAlist)))))
    opAlist
  (dolist (item opAlist)
    (dolist (pair (cdr item))
      (rplacd pair (list t conform t (second pair))))
    opAlist))
---

— initvars —

(defun initvars
  (defvar message1 (concatenate 'string
    "{\sf Record(a:A,b:B)} is used to create the class of pairs of objects 
" "made up of a value of type {\em A} selected by the symbol {\em a} and " 
"a value of type {\em B} selected by the symbol {\em b}. " 
"In general, the {\sf Record} constructor can take any number of arguments " 
"and thus can be used to create aggregates of heterogeneous components of " 
"arbitrary size selectable by name. " 
"{\sf Record} is a primitive domain of Axiom which cannot be " 
"defined in the Axiom language."))
---
--- postvars ---

(eval-when (eval load)
  (put 'Record 'documentation
    (subst message1 'message
      `((|constructor| (nil message))
        (= (((|Boolean|) $ $)
            "\spad{r = s} tests for equality of two records \spad{r} and \spad{s}")
          (|coerce| (((|OutputForm|) $)
            "\spad{coerce(r)} returns an representation of \spad{r} as an output form")
          (($ (|List| (|Any|)))
            (,concatenate 'string
              "\spad{coerce(u)}, where \spad{u} is the list \spad{[x,y]} for \spad{x} "
              "of type \spad{A} and \spad{y} of type \spad{B}, returns the record "
              "\spad{[a:x,b:y]})")
          ,,(concatenate 'string
            "\spad{a} returns the value stored in record \spad{r} under "
            "selector \spad{a}.")
          ,(B $ "b")
          ,,(concatenate 'string
            "\spad{b} returns the value stored in record \spad{r} "
            "under selector \spad{b}.")
          ,,(concatenate 'string
            "\spad{a := x} destructively replaces the value stored in "
            "record \spad{r} under selector \spad{a} by the value of \spad{x}. "
            "Error: if \spad{r} has not been previously assigned a value."))
          ,(B $ "b")
          ,,(concatenate 'string
            "\spad{b := y} destructively replaces the value stored in "
            "record \spad{r} under selector \spad{b} by the value of \spad{y}. "
            "Error: if \spad{r} has not been previously assigned a value.")))
    :test #'equal)))

--- initvars ---

(defvar message2 (concatenate 'string
  "{\sf Union(A,B)} denotes the class of objects which are which are either "
  "members of domain \{\em A\} or of domain \{\em B\}. The \{\sf Union\} "
  "constructor can take any number of arguments. "
  "For an alternate form of \{\sf Union\} with \""tags\"", see "
  "\downlink{Union(a:A,b:B)}{DomainUnion}. \{\sf Union\} is a primitive "
  "domain of Axiom which cannot be defined in the Axiom language."))
43.6. CONSTRUCTOR PAGE MENU

---

**postvars**

```
(eval-when (eval load)
  (put '|UntaggedUnion| '|documentation|
    (subst message2 'message
     ('(((|constructor| (nil message))
       (= (((|Boolean|) $ $)
           ,(|concatenate| 'string
           "|\spad{u = v} tests if two objects of the union are equal, 
             "
           "that is, u and v are hold objects of same branch which are equal.").))))
     ,(|concatenate| 'string
     "|\spad{u case A} tests if |\spad{u} is of the type \spad{A} 
       "branch of the union."))
     ,(|concatenate| 'string
     "|\spad{u case B} tests if |\spad{u} is of the \spad{B} branch 
       "of the union."))
     ,(|concatenate| 'string
     "|\spad{coerce(u)} returns |\spad{x} of type \spad{A} if 
       "|\spad{x} is of the \spad{A} branch of the union. 
       "|Error: if |\spad{u} is of the \spad{B} branch of the union.").))
     ,(|concatenate| 'string
     "|\spad{coerce(u)} returns |\spad{x} of type \spad{B} if 
       "|\spad{x} is of the \spad{B} branch of the union. 
       "|Error: if \spad{u} is of the \spad{A} branch of the union.").))
     ,(|concatenate| 'string
     "|\spad{coerce(x)}, where \spad{x} has type \spad{A}, 
       "returns \spad{x} as a union type."))
     ,(|concatenate| 'string
     "|\spad{coerce(y)}, where \spad{y} has type \spad{B}, 
       "returns \spad{y} as a union type."))
  :test #'equal))
```

---

**initvars**

```
(defvar message3 (concatenate 'string
  "\{\em Union(a:A,b:B)\} denotes the class of objects which are either 
  "members of domain \{\em A\} or of domain \{\em B\}. 
  ")
```
The symbols \emph{a} and \emph{b} are called "tags" and are used to "identify the two "branches" of the union." The \texttt{Union} constructor can take any number of arguments and has an "alternate form without \emph{tags}" (see \texttt{Union(A,B)}{UntaggedUnion}). "This tagged \texttt{Union} type is necessary, for example, to disambiguate" two branches of a union where \emph{A} and \emph{B} denote the same type. "\texttt{Union} is a primitive domain of Axiom which cannot be "defined in the Axiom language.")

——— postvars ———

(eval-when (eval load)
(put '|Union| '|documentation|
(subst message3 'message
 '(((|constructor| (NIL MESSAGE))
 (= (((|Boolean|) $ $)
 ,(|concatenate 'string
 "\spad{u = v} tests if two objects of the union are equal, that "
"is, \spad{u} and \spad{v} are objects of same branch which are equal.")))
 (|case| (((|Boolean|) $ "A")
 "\spad{u case a} tests if \spad{u} is of branch \spad{a} of the union.")
 (((|Boolean|) $ "B")
 "\spad{u case b} tests if \spad{u} is of branch \spad{b} of the union."))
 (|coerce| ((A $)
 ,(|concatenate 'string
 "\spad{coerce(u)} returns \spad{x} of type \spad{A} if "
"\spad{x} is of branch \spad{a} of the union."))
 ((B $)
 ,(|concatenate 'string
 "\spad{coerce(u)} returns \spad{x} of type \spad{B} if "
"\spad{x} is of branch \spad{b} branch of the union."))
 "Error: if \spad{u} is of branch \spad{a} branch of the union.")
 ,(|concatenate 'string
 "\spad{coerce(x)}, where \spad{x} is of type \spad{A}, returns "
"\spad{x} as a union type."))
 ,(|concatenate 'string
 "\spad{coerce(y)}, where \spad{y} is of type \spad{B}, returns "
"\spad{y} as a union type.")))))
 :test #'equal)))

———
43.6. CONSTRUCTOR PAGE MENU

— initvars —

(defvar message4 (concatenate 'string
"{\textsf{Mapping(T,S,...)}} denotes the class of objects which are mappings from "
"a source domain \{\textem S,...\} into a target domain \textem T}. The "
"{\textsf{Mapping}} constructor can take any number of arguments."
" All but the first argument is regarded as part of a source tuple for the "
"mapping. For example, \{\textsf{Mapping(T,A,B)}\} denotes the class of mappings "
"from \{\textem (A,B)\} into \textem T}. "
"{\textsf{Mapping}} is a primitive domain of Axiom which cannot be defined in "
"the Axiom language."))

— postvars —

(eval-when (eval load)
(p put '|Mapping| '|documentation|
(subst message4 'message
 '((|constructor| (NIL MESSAGE))
 (= (((|Boolean|) $ $)
   "\texttt{(u = v)} tests if mapping objects are equal.")))
:test #'equal)))

— initvars —

(defvar message5 (concatenate 'string
"{\textem Enumeration(a1, a2 , ..., aN)} creates an object which is exactly one "
"of the N symbols \{\textem a1}, \{\textem a2}, ..., or \{\textem aN}, N > 0. "
" The \{\textem Enumeration\} can constructor can take any number of symbols as "
"arguments."))

— postvars —

(eval-when (eval load)
(p put '|Enumeration| '|documentation|
(subst message5 'message
 '((|constructor| (nil message))


(= (((|Boolean|) $ $)
,(concatenate 'string
"\spad{e = f} tests for equality of two enumerations \spad{e} "
"and \spad{f}")))
(\= (((|Boolean|) $ $)
,(concatenate 'string
"\spad{e \!=\! f} tests that two enumerations \spad{e} and "
"\spad{f} are not equal")))
(|coerce| (((|OutputForm|) $)
,(concatenate 'string
"\spad{coerce(e)} returns a representation of enumeration "
"\spad{r} as an output form"))
|\$ (|Symbol|))
,(concatenate 'string
"\spad{coerce(s)} converts a symbol \spad{s} into an "
"enumeration which has \spad{r} as a member symbol"))
:test #'equal))

---

defun lefts

[hlkeys p1195]
[hascategory-hash p??]

— defun lefts —

defun |lefts| (u)
(let (keys)
 (setq keys (hkeys *hascategory-hash*))
 (loop for x in keys when (equal (cdr x) u) collect x)))

---

Build Library Database (libdb.text,...)

defun dbMkForm

— defun dbMkForm —

defun |dbMkForm| (x)
 (or (and (atom x) (cons x nil)) x)
defun libConstructorSig

(getdatabase p1156)
(take p??)
(length p??)
(sublislis p??)
(form2LispString p??)
(ncParseFromString p1240)
[sayBrightly p??]
[$TriangleVariableList p??]

— defun libConstructorSig —

(defun |libConstructorSig| (arg)
  (labels ((
    (fn (x)
      (cond ((atom x) x)
            ((and (consp x) (eq (qcar x) '|Join|) (consp (qcdr x)))
             (list '|Join| (fn (qcadr x)) '|etc|))
            ((and (consp x) (eq (qcar x) 'category))
             '|etc|)
            (t
             (loop for y in x collect (fn y))))
    (g (x u i)
      "does x appear in any but i-th element of u?"
      (some #'identity
        (loop for y in u for j from 1
          when (not (= i j))
          collect (contained x y))))
    (let (conname argl formals keys sig sigpart)
      (declare (special |$TriangleVariableList|))
      (setq conname (car arg))
      (setq argl (cdr arg))
      (setq sig (cdar (getdatabase conname 'constructormodemap)))
      (setq formals (take (|#| argl) |$FormalMapVariableList|))
      (setq sig (sublislis formals |$TriangleVariableList| sig))
      (setq keys
        (loop for f in formals for i from 1
          collect (g f sig i))
      )
      (setq sig
        (fn (sublislis argl |$FormalMapVariableList| sig)))
      (setq sig (cons (car sig)
        (loop for a in argl for s in (cdr sig) for k in keys
          collect (if k (list #\: a s) s)))
      )
      (setq sigpart (|form2LispString| (cons '|Mapping| sig))
        (unless (|ncParseFromString| sigpart)
          (|sayBrightly| (list "Won't parse: " sigpart))
        )
      ))
)
Chapter 44

Utility functions

defun readline

— defun readline —

(DEFUN READLINE (t1)
 (if t1
  (%read-line t1)
  (%read-line *STANDARD-INPUT*))))
Chapter 45

The Interpreter

— Interpreter —

(setq *print-array* nil)
(setq *print-circle* nil)
(setq *print-pretty* nil)

(in-package "BOOT")
\getchunk{initvars}

;;; level 0 macros

\getchunk{defmacro bit-to-truth 0}
\getchunk{defmacro bvec-elt 0}
\getchunk{defmacro idChar? 0}
\getchunk{defmacro identp 0}
\getchunk{defmacro frameExposureData 0}
\getchunk{defmacro frameHiFiAccess 0}
\getchunk{defmacro frameHistListAct 0}
\getchunk{defmacro frameHistList 0}
\getchunk{defmacro frameHistListLen 0}
\getchunk{defmacro frameHistoryTable 0}
\getchunk{defmacro frameHistRecord 0}
\getchunk{defmacro frameInteractive 0}
\getchunk{defmacro frameIOIndex 0}
\getchunk{defmacro frameName 0}
\getchunk{defmacro frameNames 0}
\getchunk{defmacro getMsgArgL 0}
\getchunk{defmacro getMsgKey 0}
\getchunk{defmacro getMsgPosTagOb 0}
\getchunk{defmacro getMsgPrefix 0}
\getchunk{defmacro getMsgPrefix? 0}
defmacro getMsgTag 0
defmacro getMsgTag? 0
defmacro getMsgText 0
defmacro hashCode? 0
defmacro qsabsval 0
defmacro qsadd1 0
getchunk[defmacro qsdifference 0]
defmacro qsgreaterp 0
getchunk[defmacro qslesssp 0]
defmacro qsmax 0
getchunk[defmacro qsmind 0]
defmacro qsmind 0
getchunk[defmacro qsoddp 0]
defmacro qspopl 0
getchunk[defmacro qssub1 0]
defmacro qstimes 0
getchunk[defmacro qszerop 0]
defmacro setMsgPrefix 0
defmacro setMsgText 0
getchunk[defmacro spadConstant 0]

;;;; above level 0 macros

defmacro ancolsU8
defmacro ancolsU16
defmacro ancolsU32
getchunk[defmacro anrowsU8]
defmacro anrowsU16
getchunk[defmacro anrowsU32]
defmacro aref2U8
getchunk[defmacro aref2U16]
defmacro aref2U32
getchunk[defmacro assq]
defmacro bvec-setelt
getchunk[defmacro bvec-size]
defmacro cdaref2
getchunk[defmacro cdelt]
defmacro cdlen
getchunk[defmacro cdancols]
defmacro cdanrows
getchunk[defmacro cdsetaref2]
defmacro cdsetelt
getchunk[defmacro danrows]
defmacro dancols
getchunk[defmacro daref2]
defmacro delt
getchunk[defmacro DFAdd]
defmacro DFAcos
getchunk[defmacro DFACosh]
\getchunk{defmacro DFAsin}
\getchunk{defmacro DFAsinh}
\getchunk{defmacro DFAtan}
\getchunk{defmacro DFAtan2}
\getchunk{defmacro DFAtanh}
\getchunk{defmacro DFCos}
\getchunk{defmacro DFCosh}
\getchunk{defmacro DFDivide}
\getchunk{defmacro DFEql}
\getchunk{defmacro DFExp}
\getchunk{defmacro DFExpt}
\getchunk{defmacro DFIntegerDivide}
\getchunk{defmacro DFIntegerExpt}
\getchunk{defmacro DFIntegerMultiply}
\getchunk{defmacro DFLessThan}
\getchunk{defmacro DFLog}
\getchunk{defmacro DFLogE}
\getchunk{defmacro DFMax}
\getchunk{defmacro DFMin}
\getchunk{defmacro DFMinusp}
\getchunk{defmacro DFMultiply}
\getchunk{defmacro DFSin}
\getchunk{defmacro DFSinh}
\getchunk{defmacro DFSqrt}
\getchunk{defmacro DFSqrt}
\getchunk{defmacro DFSubtract}
\getchunk{defmacro DFTan}
\getchunk{defmacro DFTanh}
\getchunk{defmacro DFTanh}
\getchunk{defmacro DFTanh}
\getchunk{defmacro DFUnaryMinus}
\getchunk{defmacro DFZeroP}
\getchunk{defmacro dlen}
\getchunk{defmacro dsetaref2}
\getchunk{defmacro dsetelt}
\getchunk{defmacro eltU8}
\getchunk{defmacro eltU16}
\getchunk{defmacro eltU32}
\getchunk{defmacro funfind}
\getchunk{defmacro hget}
\getchunk{defmacro leader?}
\getchunk{defmacro line?}
\getchunk{defmacro make-cdouble-matrix}
\getchunk{defmacro make-cdouble-vector}
\getchunk{defmacro make-double-matrix}
\getchunk{defmacro make-double-matrix}
\getchunk{defmacro make-double-matrix1}
\getchunk{defmacro make-double-matrix}
\getchunk{defmacro make-double-vector1}
\getchunk{defmacro makeMatrixU8}
\getchunk{defmacro makeMatrix1U8}
\getchunk{defmacro makeMatrixU16}
\getchunk{defmacro makeMatrix1U16}
\getchunk{defmacro makeMatrixU32}
CHAPTER 45. THE INTERPRETER

\getchunk{defmacro makeMatrix1U32}
\getchunk{defmacro mkObj}
\getchunk{defmacro mkObjCode}
\getchunk{defmacro mkObjWrap}
\getchunk{defmacro objCodeVal}
\getchunk{defmacro objCodeMode}
\getchunk{defmacro objMode}
\getchunk{defmacro objSetMode}
\getchunk{defmacro objSetVal}
\getchunk{defmacro objVal}
\getchunk{defmacro objValUnwrap}
\getchunk{defmacro qsDot26432}
\getchunk{defmacro qsDot2Mod6432}
\getchunk{defmacro qsMod6432}
\getchunk{defmacro qsMulAdd6432}
\getchunk{defmacro qsMulAddMod6432}
\getchunk{defmacro qsMul6432}
\getchunk{defmacro qsMulMod32}
\getchunk{defmacro qvlenU8}
\getchunk{defmacro qvlenU16}
\getchunk{defmacro qvlenU32}
\getchunk{defmacro Rest}
\getchunk{defmacro startsId?}
\getchunk{defmacro setAref2U8}
\getchunk{defmacro setAref2U16}
\getchunk{defmacro setAref2U32}
\getchunk{defmacro seteltU8}
\getchunk{defmacro seteltU16}
\getchunk{defmacro seteltU32}
\getchunk{defmacro toScreen?}
\getchunk{defmacro trapNumericErrors}
\getchunk{defmacro truth-to-bit}
\getchunk{defmacro while}
\getchunk{defmacro whileWithResult}

;;;; layer 0 (all common lisp)

\getchunk{defun acot 0}
\getchunk{defun acoth 0}
\getchunk{defun acsc 0}
\getchunk{defun acsch 0}
\getchunk{defun asec 0}
\getchunk{defun asech 0}
\getchunk{defun axiomVersion 0}
\getchunk{defun BooleanEquality 0}
\getchunk{defun bvec-and 0}
\getchunk{defun bvec-concat 0}
\getchunk{defun bvec-copy 0}
\getchunk{defun bvec-equal 0}
\getchunk{defun bvec-greater 0}
\getchunk{defun bvec-make-full 0}
\getchunk{defun bvec-nand 0}
\getchunk{defun bvec-nor 0}
\getchunk{defun bvec-not 0}
\getchunk{defun bvec-or 0}
\getchunk{defun bvec-xor 0}

\getchunk{defun concatWithBlanks 0}
\getchunk{defun cleanupLine 0}
\getchunk{defun clearMacroTable 0}
\getchunk{defun concat 0}
\getchunk{defun cot 0}
\getchunk{defun coth 0}
\getchunk{defun createCurrentInterpreterFrame 0}
\getchunk{defun credits 0}
\getchunk{defun csc 0}
\getchunk{defun csch 0}

\getchunk{defun dbKind 0}
\getchunk{defun dbRead 0}
\getchunk{defun Delay 0}
\getchunk{defun desiredMsg 0}
\getchunk{defun DirToString 0}
\getchunk{defun displayFrameNames 0}
\getchunk{defun divide2 0}
\getchunk{defun dqAppend 0}
\getchunk{defun dqToList 0}
\getchunk{defun dqUnit 0}

\getchunk{defun emptyInterpreterFrame 0}
\getchunk{defun endedp 0}
\getchunk{defun evalSharpOne 0}

\getchunk{defun fin 0}
\getchunk{defun findFrameInRing 0}
\getchunk{defun flatten 0}
\getchunk{defun fnameExists? 0}
\getchunk{defun fnameName 0}
\getchunk{defun fnameReadable? 0}
\getchunk{defun fnameType 0}
\getchunk{defun frameNames 0}
\getchunk{defun From 0}
\getchunk{defun FromTo 0}

\getchunk{defun get-current-directory 0}
\getchunk{defun getenviron 0}
\getchunk{defun get1 0}
\getchunk{defun getLinePos 0}
\getchunk{defun getLineText 0}
CHAPTER 45. THE INTERPRETER

(defun getMsgKey? 0)
defun getParserMacroNames 0
defun getPreStL 0
defun getspoolname 0

defun hasCorrectTarget 0
defun hasOptArgs? 0

defun ignorep 0

defun incActive? 0
defun incCommand? 0
defun incDrop 0
defun inclmsgConsole 0
defun inclmsgFinSkipped 0
defun inclmsgPrematureEOF 0
defun inclmsgCmdBug 0

defun includeIfBug 0
defun incPrefix? 0

defun init-memory-config 0
defun insertPos 0

defun integer-decode-float-denominator 0
defun integer-decode-float-exponent 0
defun integer-decode-float-sign 0
defun integer-decode-float-numerator 0

defun intloopPrefix? 0
defun isIntegerString 0

defun keyword 0
defun keyword? 0

defun lastcount 0

defun lfcomment 0
defun lferror 0
defun lffloat 0
defun lfid 0

defun lfinteger 0
defun lfnegcomment 0
defun lfinteger 0
defun lfrinteger 0
defun lspaces 0
defun lstring 0

defun libdbTrim 0

defun lnCreate 0
defun lnExtraBlanks 0
defun lnFileName? 0
defun lnGlobalNum 0
defun lnImmediate? 0

defun lnLocalNum 0

defun lnPlaceOfOrigin 0

defun lnSetGlobalNum 0
\texttt{(defun lnString 0)}
\texttt{(defun mac0Define 0)}
\texttt{(defun mac0InfiniteExpansion,name 0)}
\texttt{(defun make-absolute-filename 0)}
\texttt{(defun makeByteWordVec2 0)}
\texttt{(defun makeInitialModemapFrame 0)}
\texttt{(defun manexp 0)}
\texttt{(defun member 0)}
\texttt{(defun mkObjFn 0)}
\texttt{(defun monitor-add 0)}
\texttt{(defun monitor-apropos 0)}
\texttt{(defun monitor-autoload 0)}
\texttt{(defun monitor-checkpoint 0)}
\texttt{(defun monitor-decr 0)}
\texttt{(defun monitor-delete 0)}
\texttt{(defun monitor-dirmname 0)}
\texttt{(defun monitor-disable 0)}
\texttt{(defun monitor-enable 0)}
\texttt{(defun monitor-end 0)}
\texttt{(defun monitor-exposedp 0)}
\texttt{(defun monitor-file 0)}
\texttt{(defun monitor-help 0)}
\texttt{(defun monitor-incr 0)}
\texttt{(defun monitor-info 0)}
\texttt{(defun monitor-inittable 0)}
\texttt{(defun monitor-libname 0)}
\texttt{(defun monitor-nrllib 0)}
\texttt{(defun monitor-parse 0)}
\texttt{(defun monitor-percent 0)}
\texttt{(defun monitor-readinterp 0)}
\texttt{(defun monitor-report 0)}
\texttt{(defun monitor-reset 0)}
\texttt{(defun monitor-restore 0)}
\texttt{(defun monitor-results 0)}
\texttt{(defun monitor-spadfile 0)}
\texttt{(defun monitor-tested 0)}
\texttt{(defun monitor-untested 0)}
\texttt{(defun monitor-write 0)}
\texttt{(defun ncError 0)}
\texttt{(defun ncloopEscaped 0)}
\texttt{(defun ncloopPrefix? 0)}
\texttt{(defun ncloopPrintLines 0)}
\texttt{(defun nonBlank 0)}
\texttt{(defun npAnyNo 0)}
\texttt{(defun npboot 0)}
\texttt{(defun npEqPeek 0)}
\texttt{(defun npPop1 0)}
\texttt{(defun npPop2 0)}
\texttt{(defun npPop3 0)}
(defun pfParts 0)
(defun pfPile 0)
(defun pfPretendExpr 0)
(defun pfPretendType 0)
(defun pfRestrictExpr 0)
(defun pfRestrictType 0)
(defun pfReturnExpr 0)
(defun pfRuleLhsItems 0)
(defun pfRuleRhs 0)
(defun pfSecond 0)
(defun pfSequenceArgs 0)
(defun pfSuchthatCond 0)
(defun pfTaggedExpr 0)
(defun pfTaggedTag 0)
(defun pfTree 0)
(defun pfTypedId 0)
(defun pfTypedType 0)
(defun pfTupleParts 0)
(defun pfWhereContext 0)
(defun pfWhereExpr 0)
(defun pfWhileCond 0)
(defun placep 0)
(defun pmDontQuote? 0)
(defun poCharPosn 0)
(defun poGetLineObject 0)
(defun poNoPos? 0)
(defun poNoPosition 0)
(defun poNoPosition? 0)
(defun printAsTeX 0)
(defun pname 0)

(defun qenum 0)
(defun qeset 0)
(defun qaquotient 0)
(defun qremainder 0)
(defun quotient2 0)

(defun random 0)
(defun rdigit? 0)
(defun reclaim 0)
(defun remainder2 0)
(defun remLine 0)
(defun rep 0)
(defun resetStackLimits 0)
(defun resultp 0)

(defun sameUnionBranch 0)
(defun satisfiesUserLevel 0)
(defun scanCloser? 0)
(defun sec 0)
CHAPTER 45. THE INTERPRETER

\getchunk{defun sech 0}
\getchunk{defun setCurrentLine 0}
\getchunk{defun set-restart-hook 0}
\getchunk{defun showMsgPos? 0}
\getchunk{defun startsComment? 0}
\getchunk{defun StreamNull 0}
\getchunk{defun stringPrefix? 0}
\getchunk{defun stripLisp 0}
\getchunk{defun stripSpaces 0}
\getchunk{defun substring 0}
\getchunk{defun theid 0}
\getchunk{defun thefname 0}
\getchunk{defun theorigin 0}
\getchunk{defun tokPart 0}
\getchunk{defun To 0}
\getchunk{defun Top? 0}
\getchunk{defun trademark 0}
\getchunk{defun vmread 0}
\getchunk{defun zeroOneTran 0}

;;; above level 0
\getchunk{defun abbQuery}
\getchunk{defun abbreviations}
\getchunk{defun abbreviationsSpad2Cmd}
\getchunk{defun absolutelyCanCoerceByCheating}
\getchunk{defun addBinding}
\getchunk{defun addBindingInteractive}
\getchunk{defun addInputLibrary}
\getchunk{defun addNewInterpreterFrame}
\getchunk{defun addOperations}
\getchunk{defun addTraceItem}
\getchunk{defun algCoerceInteractive}
\getchunk{defun algEqual}
\getchunk{defun allConstructors}
\getchunk{defun allOperations}
\getchunk{defun alqlGetOrigin}
\getchunk{defun alqlGetParams}
\getchunk{defun alqlGetKindString}
\getchunk{defun alreadyOpened?}
\getchunk{defun apropos}
\getchunk{defun assertCond}
\getchunk{defun augmentHasArgs}
\getchunk{defun augmentTraceNames}

\getchunk{defun basicLookup}
\getchunk{defun basicLookupCheckDefaults}
\getchunk{defun basicStringize} 
\getchunk{defun bcComplexLimit} 
\getchunk{defun bcComplexLimitGen} 
\getchunk{defun bcCreateVariableString} 
\getchunk{defun bcDefiniteIntegrate} 
\getchunk{defun bcDefiniteIntegrateGen} 
\getchunk{defun bcDifferentiate} 
\getchunk{defun bcDifferentiateGen} 
\getchunk{defun bcDraw} 
\getchunk{defun bcDrawGen} 
\getchunk{defun bcDrawIt} 
\getchunk{defun bcDrawIt2} 
\getchunk{defun bcDraw2Dfunc} 
\getchunk{defun bcDraw2DfuncGen} 
\getchunk{defun bcDraw2Dpar} 
\getchunk{defun bcDraw2DparGen} 
\getchunk{defun bcDraw2DSolve} 
\getchunk{defun bcDraw2DSolveGen} 
\getchunk{defun bcDraw3Dfunc} 
\getchunk{defun bcDraw3DfuncGen} 
\getchunk{defun bcDraw3Dpar} 
\getchunk{defun bcDraw3DparGen} 
\getchunk{defun bcDraw3Dpar1} 
\getchunk{defun bcDraw3Dpar1Gen} 
\getchunk{defun bcError} 
\getchunk{defun bcFindString} 
\getchunk{defun bcFinish} 
\getchunk{defun bcGen} 
\getchunk{defun bcGenEquations} 
\getchunk{defun bcGenExplicitMatrix} 
\getchunk{defun bcHt} 
\getchunk{defun bchtMakeButton} 
\getchunk{defun bcIndefiniteIntegrate} 
\getchunk{defun bcIndefiniteIntegrateGen} 
\getchunk{defun bcInputEquations} 
\getchunk{defun bcInputEquationsEnd} 
\getchunk{defun bcInputExplicitMatrix} 
\getchunk{defun bcInputMatrixByFormula} 
\getchunk{defun bcInputMatrixByFormulaGen} 
\getchunk{defun bcInputSolveInfo} 
\getchunk{defun bcIssueHt} 
\getchunk{defun bcLaurentSeries} 
\getchunk{defun bcLaurentSeriesGen} 
\getchunk{defun bcLimit} 
\getchunk{defun bcLinearExtractMatrix} 
\getchunk{defun bcLinearMatrixGen} 
\getchunk{defun bcLinearSolve} 
\getchunk{defun bcLinearSolveEqns} 
\getchunk{defun bcLinearSolveEqns1} 
\getchunk{defun bcLinearSolveEqnsGen} 
\getchunk{defun bcLinearSolveMatrixGen}
CHAPTER 45. THE INTERPRETER

\getchunk{defun bcLinearSolveMatrix1}
\getchunk{defun bcLinearSolveMatrixHomo}
\getchunk{defun bcLinearSolveMatrixInhomo}
\getchunk{defun bcLinearSolveMatrixInhomoGen}
\getchunk{defun bcMatrix}
\getchunk{defun bcMatrixGen}
\getchunk{defun bcMakeEquations}
\getchunk{defun bcMakeLinearEquations}
\getchunk{defun bcMakeUnknowns}
\getchunk{defun bcMkFunction}
\getchunk{defun bcNotReady}
\getchunk{defun bcOptional}
\getchunk{defun bcProduct}
\getchunk{defun bcProductGen}
\getchunk{defun bcPuiseuxSeries}
\getchunk{defun bcPuiseuxSeriesGen}
\getchunk{defun bcReadMatrix}
\getchunk{defun bcRealLimit}
\getchunk{defun bcRealLimitGen}
\getchunk{defun bcRealLimitGen1}
\getchunk{defun bcSadFaces}
\getchunk{defun bcSeries}
\getchunk{defun bcSeriesByFormula}
\getchunk{defun bcSeriesByFormulaGen}
\getchunk{defun bcSeriesExpansion}
\getchunk{defun bcSeriesExpansionGen}
\getchunk{defun bcSeriesGen}
\getchunk{defun bcSolve}
\getchunk{defun bcSolveEquations}
\getchunk{defun bcSolveEquationsNumerically}
\getchunk{defun bcSolveNumerically1}
\getchunk{defun bcSolveSingle}
\getchunk{defun bcString2HyString}
\getchunk{defun bcString2HyString2}
\getchunk{defun bcString2WordList}
\getchunk{defun bcSystemSolveEqns1}
\getchunk{defun bcSum}
\getchunk{defun bcSumGen}
\getchunk{defun bcSystemSolve}
\getchunk{defun bcTaylorSeries}
\getchunk{defun bcTaylorSeriesGen}
\getchunk{defun bcUnixTable}
\getchunk{defun bcVectorGen}
\getchunk{defun bcvspace}
\getchunk{defun bcwords2liststring}
\getchunk{defun beforeAfter}
\getchunk{defun bracketString}
\getchunk{defun break}
\getchunk{defun breaklet}
\getchunk{defun brightprint}
(defun brightprint-0)
(defun browse)
(defun browseopen)
(defun buildHtMacroTable)
(defun buttonNames)

(defun canFuncall?)
(defun categoryopen)
(defun catchCoerceFailure)
(defun changeHistListLen)
(defun changeToNamedInterpreterFrame)
(defun charDigitVal)
(defun checkCondition)
(defun checkFilter)
(defun chkAllNonNegativeInteger)
(defun chkDirectory)
(defun chkNameList)
(defun chkNonNegativeInteger)
(defun chkOutputFileName)
(defun chkPosInteger)
(defun chkRange)
(defun cleanline)
(defun clear)
(defun clearCmdAll)
(defun clearCmdCompletely)
(defun clearCmdExcept)
(defun clearCmdParts)
(defun clearCmdSortedCaches)
(defun clearFrame)
(defun clearParserMacro)
(defun clearSpad2Cmd)
(defun close)
(defun closeInterpreterFrame)
(defun cmpnote)
(defun coerceBranch2Union)
(defun coerceByFunction)
(defun coerceByTable)
(defun coerceCommutetest)
(defun coerceConvertMmSelection)
(defun coerceImmediateSubDomain)
(defun coerceInt)
(defun coerceInt0)
(defun coerceInt1)
(defun coerceIntX)
(defun coerceIntAlgebraicConstant)
(defun coerceIntByMap)
(defun coerceIntByMapInner)
(defun coerceIntCommutate)
(defun coerceInteractive)
(defun coerceIntFromUnion)
\getchunk{defun coerceIntPermute}
\getchunk{defun coerceIntSpecial}
\getchunk{defun coerceIntTableOrFunction}
\getchunk{defun coerceIntTest}
\getchunk{defun coerceIntTower}
\getchunk{defun coerceInt2Union}
\getchunk{defun coerceOrRetract}
\getchunk{defun coerceOrThrowFailure}
\getchunk{defun coerceSpadArgs2E}
\getchunk{defun coerceSpadFunValue2E}
\getchunk{defun coerceSubDomain}
\getchunk{defun coerceTraceArgs2E}
\getchunk{defun coerceTraceFunValue2E}
\getchunk{defun coerceUnion2Branch}
\getchunk{defun coercionFailure}
\getchunk{defun commandAmbiguityError}
\getchunk{defun commandError}
\getchunk{defun commandErrorIfAmbiguous}
\getchunk{defun commandErrorMessage}
\getchunk{defun commandsForUserLevel}
\getchunk{defun commandUserLevelError}
\getchunk{defun compareposns}
\getchunk{defun compareTypeLists}
\getchunk{defun compileBoot}
\getchunk{defun compiledLookup}
\getchunk{defun compiledLookupCheck}
\getchunk{defun computeDomainVariableAlist}
\getchunk{defun computeTTTranspositions}
\getchunk{defun condErrorMsg}
\getchunk{defun conLowerCaseConTran}
\getchunk{defun conOpPage}
\getchunk{defun conOpPage1}
\getchunk{defun conPage}
\getchunk{defun conPageChoose}
\getchunk{defun conPageConEntry}
\getchunk{defun conPageFastPath}
\getchunk{defun conSpecialString?}
\getchunk{defun constoken}
\getchunk{defun constructorSearch}
\getchunk{defun constructSubst}
\getchunk{defun containsVars}
\getchunk{defun containsVars1}
\getchunk{defun copyright}
\getchunk{defun countCache}
\getchunk{defun cSearch}

\getchunk{defun DaaseName}
\getchunk{defun dbAddChain}
\getchunk{defun dbAddChainDomain}
\getchunk{defun dbAddDocTable}
\getchunk{defun dbCompositeWithMap}
\getchunk{defun dbConsExposureMessage}
\getchunk{defun dbConsHeading}
\getchunk{defun dbConstructorDoc}
\getchunk{defun dbConstructorDoc,hn}
\getchunk{defun dbConstructorDoc,gn}
\getchunk{defun dbDocTable}
\getchunk{defun dbExtractUnderlyingDomain}
\getchunk{defun dbGetDocTable}
\getchunk{defun dbGetDocTable,gn}
\getchunk{defun dbGetDocTable,hn}
\getchunk{defun dbMkForm}
\getchunk{defun dbNonEmptyPattern}
\getchunk{defun dbSearchOrder}
\getchunk{defun dbSelectCon}
\getchunk{defun dbShowConditions}
\getchunk{defun dbShowCons}
\getchunk{defun dbShowCons1}
\getchunk{defun dbShowConsDoc}
\getchunk{defun dbShowConsDoc1}
\getchunk{defun dbShowConsKindsFilter}
\getchunk{defun dbShowConstructorLines}
\getchunk{defun dbSpecialDescription}
\getchunk{defun dbSpecialExpandIfNecessary}
\getchunk{defun dbSpecialExports}
\getchunk{defun dbSpecialOperations}
\getchunk{defun dbString2Words}
\getchunk{defun dbSubConform}
\getchunk{defun dbWordFrom}
\getchunk{defun decideHowMuch}
\getchunk{defun decomposeTypeIntoTower}
\getchunk{defun defaultTargetFE}
\getchunk{defun defiostream}
\getchunk{defun deldatabase}
\getchunk{defun deleteFile}
\getchunk{defun describe}
\getchunk{defun describeFortPersistence}
\getchunk{defun describeInputLibraryArgs}
\getchunk{defun describeOutputLibraryArgs}
\getchunk{defun describeSetFortDir}
\getchunk{defun describeSetFortTmpDir}
\getchunk{defun describeSetFunctionsCache}
\getchunk{defun describeSetLinkerArgs}
\getchunk{defun describeSetNagHost}
\getchunk{defun describeSetOutputAlgebra}
\getchunk{defun describeSetOutputFormula}
\getchunk{defun describeSetOutputFortran}
\getchunk{defun describeSetOutputHtml}
\getchunk{defun describeSetOutputMathml}
\getchunk{defun describeSetOutputOpenMath}
\getchunk{defun describeSetOutputTex}
\getchunk{defun describeSetStreamsCalculate}
\getchunk{defun describeSpad2Cmd}
\getchunk{defun dewritify}
\getchunk{defun dewritify,dewritifyInner}
\getchunk{defun diffAlist}
\getchunk{defun digit?}
\getchunk{defun digitp}
\getchunk{defun disableHist}
\getchunk{defun display}
\getchunk{defun displayCondition}
\getchunk{defun displayExposedConstructors}
\getchunk{defun displayExposedGroups}
\getchunk{defun displayHiddenConstructors}
\getchunk{defun displayMacro}
\getchunk{defun displayMacros}
\getchunk{defun displayMode}
\getchunk{defun displayModemap}
\getchunk{defun displayOperations}
\getchunk{defun displayOperationsFromLisplib}
\getchunk{defun displayParserMacro}
\getchunk{defun displayProperties}
\getchunk{defun displayProperties,sayFunctionDeps}
\getchunk{defun displaySetOptionInformation}
\getchunk{defun displaySetVariableSettings}
\getchunk{defun displaySpad2Cmd}
\getchunk{defun displayType}
\getchunk{defun displayValue}
\getchunk{defun displayWorkspaceNames}
\getchunk{defun doDoitButton}
\getchunk{defun domainDescendantsOf}
\getchunk{defun domainToGenvar}
\getchunk{defun domArg}
\getchunk{defun domArg2}
\getchunk{defun doSystemCommand}
\getchunk{defun downcase}
\getchunk{defun downlink}
\getchunk{defun dqConcat}
\getchunk{defun dropInputLibrary}
\getchunk{defun dSearch}
\getchunk{defun dumbTokenize}

\getchunk{defun edit}
\getchunk{defun editFile}
\getchunk{defun editSpad2Cmd}
\getchunk{defun Else?}
\getchunk{defun Elseif?}
\getchunk{defun enPile}
\getchunk{defun eofp}
\getchunk{defun eqpileTree}
\getchunk{defun erMsgCompare}
\getchunk{defun erMsgSep}
\getchunk{defun erMsgSort}
\getchunk{defun evalCategory}
\getchunk{defun evalDomain}
\getchunk{defun evaluateSignature}
\getchunk{defun evaluateType}
\getchunk{defun evaluateType1}
\getchunk{defun executeInterpreterCommand}
\getchunk{defun ExecuteInterpSystemCommand}
\getchunk{defun executeQuietCommand}
\getchunk{defun explainLinear}
\getchunk{defun fetchOutput}
\getchunk{defun fillerSpaces}
\getchunk{defun filterAndFormatConstructors}
\getchunk{defun filterListOfStrings}
\getchunk{defun filterListOfStringsWithFn}
\getchunk{defun finalExactRequest}
\getchunk{defun findnexttest}
\getchunk{defun firstTokPosn}
\getchunk{defun fixObjectForPrinting}
\getchunk{defun flattenOperationAlist}
\getchunk{defun float2Sex}
\getchunk{defun fnameDirectory}
\getchunk{defun fnameMake}
\getchunk{defun fnameNew}
\getchunk{defun fnameWritable?}
\getchunk{defun frame}
\getchunk{defun frameEnvironment}
\getchunk{defun frameSpad2Cmd}
\getchunk{defun funfind,LAM}
\getchunk{defun gatherGlossLines}
\getchunk{defun genDomainTraceName}
\getchunk{defun gensymInt}
\getchunk{defun getAliasIfTracedMapParameter}
\getchunk{defun getAndEvalConstructorArgument}
\getchunk{defun getAndSay}
\getchunk{defun getBpiNameIfTracedMap}
\getchunk{defun getBrowseDatabase}
\getchunk{defun getConstantFromDomain}
\getchunk{defun getConstructorDocumentation}
\getchunk{defun getdatabase}
\getchunk{defun getDependentsOfConstructor}
\getchunk{defun getDirectoryList}
\getchunk{defun getFirstWord}
\getchunk{defun getHtMacroItem}
\getchunk{defun getMapSig}
\getchunk{defun getMapSubNames}
\getchunk{defun getMsgCatAttr}
\getchunk{defun getMsgFTTag?}
\getchunk{defun getMsgInfoFromKey}
\getchunk{defun getMsgPos}
\getchunk{defun getMsgPos2}
\getchunk{defun getMsgToWhere}
\getchunk{defun getOplistForConstructorForm}
\getchunk{defun getOplistWithUniqueSignatures}
\getchunk{defun getOption}
\getchunk{defun getPosStL}
\getchunk{defun getPreviousMapSubNames}
\getchunk{defun getProplist}
\getchunk{defun getRefvU8}
\getchunk{defun getRefvU16}
\getchunk{defun getRefvU32}
\getchunk{defun getStFromMsg}
\getchunk{defun getSubDomainPredicate}
\getchunk{defun getSystemCommandLine}
\getchunk{defun getTraceOption}
\getchunk{defun getTraceOption,hn}
\getchunk{defun getTraceOptions}
\getchunk{defun getUsersOfConstructor}
\getchunk{defun getWorkspaceNames}

\getchunk{defun handleNoParseCommands}
\getchunk{defun handleParsedSystemCommands}
\getchunk{defun handleTokenizeSystemCommands}
\getchunk{defun hasAtt}
\getchunk{defun hasAttSig}
\getchunk{defun hasCatExpression}
\getchunk{defun hasCate}
\getchunk{defun hasCateSpecial}
\getchunk{defun hasCateSpecialNew}
\getchunk{defun hasCatei}
\getchunk{defun hasCaty}
\getchunk{defun hasCaty1}
\getchunk{defun hashashable}
\getchunk{defun hasOption}
\getchunk{defun hasPair}
\getchunk{defun hasSig}
\getchunk{defun hasSigAnd}
\getchunk{defun hasSigOr}
\getchunk{defun help}
\getchunk{defun helpSpad2Cmd}
\getchunk{defun histFileErase}
\getchunk{defun histFileName}
\getchunk{defun histInputFileName}
\getchunk{defun history}
\getchunk{defun historySpad2Cmd}
\getchunk{defun hkeys}
\getchunk{defun hput}
\getchunk{defun htAddHeading}
\getchunk{defun htAllOrNum}
\getchunk{defun htBcLinks}
\getchunk{defun htBcLispLinks}
\getchunk{defun htBcRadioButtons}
\getchunk{defun htCacheAddChoice}
\getchunk{defun htCacheOne}
\getchunk{defun htCacheSet}
\getchunk{defun htCheckList}
\getchunk{defun htCheck}
\getchunk{defun htDoneButton}
\getchunk{defun htDoNothing}
\getchunk{defun htEscapeString}
\getchunk{defun htFunctionSetLiteral}
\getchunk{defun htGlossPage}
\getchunk{defun htGloss}
\getchunk{defun htGreekSearch}
\getchunk{defun htInitPage}
\getchunk{defun htInputStrings}
\getchunk{defun htKill}
\getchunk{defun htLispLinks}
\getchunk{defun htLispMemoLinks}
\getchunk{defun htMakeButton}
\getchunk{defun htMakeDoitButton}
\getchunk{defun htMakeDoneButton}
\getchunk{defun htMakeErrorPage}
\getchunk{defun htMakeInputList}
\getchunk{defun htMakeLabel}
\getchunk{defun htMakePage}
\getchunk{defun htMakePage1}
\getchunk{defun htMakePathKey,fnt}
\getchunk{defun htMakePathKey}
\getchunk{defun htMakeTemplates,substLabel}
\getchunk{defun htMakeTemplates}
\getchunk{defun htMarkTree}
\getchunk{defun htMkName}
\getchunk{defun htpAddInputAreaProp}
\getchunk{defun htpAddToPageDescription}
\getchunk{defun htpButtonValue}
\getchunk{defun htpDestroyPage}
\getchunk{defun htpDomainConditions}
\getchunk{defun htpDomainPvarSubstList}
\getchunk{defun htpDomainVariableAlist}
\getchunk{defun htpInputAreaAlist}
\getchunk{defun htpLabelDefault}
\getchunk{defun htpLabelErrorMsg}
\getchunk{defun htpLabelOne}
\getchunk{defun htpLabelFilteredInputString}
\getchunk{defun htpLabelFilter}
\getchunk{defun htpLabelInputString}
\getchunk{defun htpLabelSpadType}
\getchunk{defun htpLabelSpadValue}
\getchunk{defun htpLabelType}
\getchunk{defun htpName}
\getchunk{defun htpPageDescription}
\getchunk{defun htpProperty}
\getchunk{defun htpPropertyList}
\getchunk{defun htProcessBcButtons}
\getchunk{defun htProcessBcStrings}
\getchunk{defun htProcessDoitButton}
\getchunk{defun htProcessDomainConditions}
\getchunk{defun htProcessDoneButton}
\getchunk{defun htProcessToggleButtons}
\getchunk{defun htSetDomainConditions}
\getchunk{defun htSetDomainPvarSubstList}
\getchunk{defun htSetDomainVariableAlist}
\getchunk{defun htSetInputAreaAlist}
\getchunk{defun htSetLabelErrorMsg}
\getchunk{defun htSetLabelInputString}
\getchunk{defun htSetLabelSpadValue}
\getchunk{defun htSetName}
\getchunk{defun htSetPageDescription}
\getchunk{defun htSetProperty}
\getchunk{defun htSetRadioButtonAlist}
\getchunk{defun htQuote}
\getchunk{defun htSetCache}
\getchunk{defun htSetExpose}
\getchunk{defun htSetFunCommandContinue}
\getchunk{defun htSetFunCommand}
\getchunk{defun htSetHistory}
\getchunk{defun htSetInputLibrary}
\getchunk{defun htSetInteger}
\getchunk{defun htSetLinkerArgs}
\getchunk{defun htSetLiterals}
\getchunk{defun htSetLiteral}
\getchunk{defun htSetNotAvailable}
\getchunk{defun htSetOutputCharacters}
\getchunk{defun htSetOutputLibrary}
\getchunk{defun htSetSystemVariableKind}
\getchunk{defun htSetSystemVariable}
\getchunk{defun htSetVars}
\getchunk{defun htsvarDoneButton}
\getchunk{defun htsShowCount}
\getchunk{defun htsShowFunctionPageContinued}
\getchunk{defun htsShowFunctionPage}
\getchunk{defun htsShowIntegerPage}
\getchunk{defun htsShowLiteralsPage}
\getchunk{defun htShowPage}
\getchunk{defun htShowPageNoScroll}
\getchunk{defun htShowSetPage}
\getchunk{defun htShowSetTreeValue}
\getchunk{defun htShowSetTree}
\getchunk{defun htStringLength}
\getchunk{defun htSystemVariables,displayOptions}
\getchunk{defun htSystemVariables,fn}
\getchunk{defun htSystemVariables,functionTail}
\getchunk{defun htSystemVariables,gn}
\getchunk{defun htSystemVariables}
\getchunk{defun htTextSearch}
\getchunk{defun htTutorialSearch}
\getchunk{defun If?}
\getchunk{defun ifCond}
\getchunk{defun iht}
\getchunk{defun importFromFrame}
\getchunk{defun incAppend}
\getchunk{defun incAppend1}
\getchunk{defun incBiteOff}
\getchunk{defun incClassify}
\getchunk{defun incCommandTail}
\getchunk{defun incConsoleInput}
\getchunk{defun incFileInput}
\getchunk{defun incFileName}
\getchunk{defun incIgen}
\getchunk{defun incIgen1}
\getchunk{defun incIgen2}
\getchunk{defun incStream}
\getchunk{defun incString}
CHAPTER 45. THE INTERPRETER

\begin{verbatim}
\getchunk{defun incZip}
\getchunk{defun incZip1}
\getchunk{defun init-boot/spad-reader}
\getchunk{defun initHist}
\getchunk{defun initHistList}
\getchunk{defun initial-getdatabase}
\getchunk{defun initializeInterpreterFrameRing}
\getchunk{defun initializeSetVariables}
\getchunk{defun initImPr}
\getchunk{defun initroot}
\getchunk{defun initToWhere}
\getchunk{defun insertAlist}
\getchunk{defun insertpile}
\getchunk{defun InterpExecuteSpadSystemCommand}
\getchunk{defun interpFunctionDepAlists}
\getchunk{defun interpopen}
\getchunk{defun interpret}
\getchunk{defun interpret1}
\getchunk{defun interpret2}
\getchunk{defun interpretTopLevel}
\getchunk{defun intInterpretPform}
\getchunk{defun intloop}
\getchunk{defun intloopEchoParse}
\getchunk{defun intloopInclude}
\getchunk{defun intloopInclude0}
\getchunk{defun intlooplisp}
\getchunk{defun intloopProcess}
\getchunk{defun intloopProcessString}
\getchunk{defun intloopReadConsole}
\getchunk{defun intloopSpadProcess}
\getchunk{defun intloopSpadProcess,interp}
\getchunk{defun intProcessSynonyms}
\getchunk{defun ioclear}
\getchunk{defun iostat}
\getchunk{defun isDomainOrPackage}
\getchunk{defun isDomainValuedVariable}
\getchunk{defun isEqualOrSubDomain}
\getchunk{defun isExposedConstructor}
\getchunk{defun isgenvar}
\getchunk{defun isInterpOnlyMap}
\getchunk{defun isListOfIdentifiers}
\getchunk{defun isListOfIdentifiersOrStrings}
\getchunk{defun isPartialMode}
\getchunk{defun isPatternVar}
\getchunk{defun isSharpVar}
\getchunk{defun isSharpVarWithNum}
\getchunk{defun isSubForRedundantMapName}
\getchunk{defun isSystemDirectory}
\getchunk{defun isTaggedUnion}
\getchunk{defun isTraceGensym}
\end{verbatim}
\defun{isUncompiledMap}
\defun{justifyMyType}
\defun{kArgPage}
\defun{kArgumentCheck}
\defun{kcaPage}
\defun{kcaPage1}
\defun{kccPage}
\defun{kcdPage}
\defun{kcdPage}
\defun{kCheckArgumentNumbers}
\defun{kcnPage}
\defun{kcPage}
\defun{kcpPage}
\defun{kDomainName}
\defun{kdPageInfo}
\defun{KeepPart?}
\defun{kePage}
\defun{kePageDisplay}
\defun{kePageOpAlist}
\defun{kePage}
\defun{kisIsValidType}
\defun{koaPageFilterByName}
\defun{koPage}
\defun{koPageAux}
\defun{koPageAux1}
\defun{koPageFromKKPage}
\defun{koPageInputAreaUnchanged?}
\defun{ksPage}
\defun{kcuPage}
\defun{kPage}
\defun{kSearch}
\defun{kTestPred}
\defun{lassocSub}
\defun{lastTokPosn}
\defun{leaveScratchpad}
\defun{lefts}
\defun{letPrint}
\defun{letPrint2}
\defun{letPrint3}
\defun{lfkey}
\defun{libConstructorSig}
\defun{library}
\defun{license}
\defun{linearFinalRequest}
\defun{lineoftoks}
\defun{linkGen}
\getchunk{defun listConstructorAbbreviations}
\getchunk{defun listDecideHowMuch}
\getchunk{defun listOfStrings2String}
\getchunk{defun listOutputter}
\getchunk{defun lnFileName}
\getchunk{defun load}
\getchunk{defun loadFunctor}
\getchunk{defun loadLib}
\getchunk{defun loadLibNoUpdate}
\getchunk{defun loadLib noLocalDatabase}
\getchunk{defun loadlnrLib}
\getchunk{defun lookupInDomainVector}
\getchunk{defun loopIters2Sex}
\getchunk{defun lotsof}
\getchunk{defun ltrace}

\getchunk{defun macApplication}
\getchunk{defun macExpand}
\getchunk{defun macId}
\getchunk{defun macLambda}
\getchunk{defun macLambda,mac}
\getchunk{defun macLambdaParameterHandling}
\getchunk{defun macMacro}
\getchunk{defun mac SUBSTITUTEId}
\getchunk{defun mac SUBSTITUTEOuter}
\getchunk{defun macroExpanded}
\getchunk{defun macro WHERE}
\getchunk{defun macro WHERE,mac}
\getchunk{defun macro EXPANDBody}
\getchunk{defun macro تقديم}
\getchunk{defun macro GETName}
\getchunk{defun macro InfiniteExpansion}
\getchunk{defun macro ModuleLambdaApply}
\getchunk{defun macro SubstituteOuter}
\getchunk{defun macro make-appendstream}
\getchunk{defun macro make-databases}
\getchunk{defun macro make-fullnamestring}
\getchunk{defun macro makeHistFile Name}
\getchunk{defun macro make InputFilename}
\getchunk{defun macro make-instream}
\getchunk{defun macro makeLeaderMsg}
\getchunk{defun macro makeMsgFromLine}
\getchunk{defun macro makeOrdinal}
\getchunk{defun macro make-outstream}
\getchunk{defun macro makePathname}
\getchunk{defun macro makeSpadCommand}
\getchunk{defun macro makeStream}
\getchunk{defun mapLetPrint}
\getchunk{defun mapStringize}
\getchunk{defun mergePathnames}
\getchunk{defun messageprint}
\getchunk{defun messageprint-1}
\getchunk{defun messageprint-2}
\getchunk{defun mkConform}
\getchunk{defun mkCurryFun}
\getchunk{defun mkDomPvar}
\getchunk{defun mkDomTypeForm}
\getchunk{defun mkEvalable}
\getchunk{defun mkEvalableMapping}
\getchunk{defun mkEvalableRecord}
\getchunk{defun mkEvalableUnion}
\getchunk{defun mkLineList}
\getchunk{defun mkprompt}
\getchunk{defun mkSetTitle}
\getchunk{defun mkUnixPattern}
\getchunk{defun msgCreate}
\getchunk{defun msgImPr?}
\getchunk{defun msgNoRep?}
\getchunk{defun msgOutputer}
\getchunk{defun msgText}
\getchunk{defun myWritable?}

\getchunk{defun namestring}
\getchunk{defun ncAlist}
\getchunk{defun ncBug}
\getchunk{defun ncConversationPhase}
\getchunk{defun ncConversationPhase,wrapup}
\getchunk{defun ncEltsQ}
\getchunk{defun ncHardError}
\getchunk{defun ncIntLoop}
\getchunk{defun ncloopCommand}
\getchunk{defun ncloopDQlines}
\getchunk{defun ncloopIncFileName}
\getchunk{defun ncloopInclude}
\getchunk{defun ncloopInclude0}
\getchunk{defun ncloopInclude1}
\getchunk{defun ncloploopParse}
\getchunk{defun ncParseFromString}
\getchunk{defun ncPutQ}
\getchunk{defun ncSoftError}
\getchunk{defun ncTag}
\getchunk{defun ncTopLevel}
\getchunk{defun newHelpSpad2Cmd}
\getchunk{defun next}
\getchunk{defun nextInterpreterFrame}
\getchunk{defun nextInterpreterFrame}
\getchunk{defun nextline}
\getchunk{defun next-lines-clear}
\getchunk{defun next-lines-show}
\getchunk{defun npAdd}
CHAPTER 45. THE INTERPRETER

\getchunk{defun npADD}
\getchunk{defun npAmpersand}
\getchunk{defun npAmpersandFrom}
\getchunk{defun npAndOr}
\getchunk{defun npAngleBared}
\getchunk{defun npApplication}
\getchunk{defun npApplication2}
\getchunk{defun npArith}
\getchunk{defun npAssign}
\getchunk{defun npAssignment}
\getchunk{defun npAssignVariable}
\getchunk{defun npAtom1}
\getchunk{defun npAtom2}
\getchunk{defun npBacksetElse}
\getchunk{defun npBackTrack}
\getchunk{defun npBDefinition}
\getchunk{defun npBPileDefinition}
\getchunk{defun npBraced}
\getchunk{defun npBracketed}
\getchunk{defun npBreak}
\getchunk{defun npBy}
\getchunk{defun npCategory}
\getchunk{defun npCategoryL}
\getchunk{defun npCoerceTo}
\getchunk{defun npColon}
\getchunk{defun npColonQuery}
\getchunk{defun npComma}
\getchunk{defun npCommaBackSet}
\getchunk{defun npCompMissing}
\getchunk{defun npConditional}
\getchunk{defun npConditionalStatement}
\getchunk{defun npConstTok}
\getchunk{defun npDDInfKey}
\getchunk{defun npDecl}
\getchunk{defun npDef}
\getchunk{defun npDefaultDecl}
\getchunk{defun npDefaultItem}
\getchunk{defun npDefaultItemlist}
\getchunk{defun npDefaultValue}
\getchunk{defun npDefinition}
\getchunk{defun npDefinitionItem}
\getchunk{defun npDefinitionlist}
\getchunk{defun npDefinitionOrStatement}
\getchunk{defun npDefn}
\getchunk{defun npDefTail}
\getchunk{defun npDiscrim}
\getchunk{defun npDisjand}
\getchunk{defun npDollar}
\getchunk{defun npDotted}
\getchunk{defun npMoveTo}
\getchunk{defun npName}
\getchunk{defun npNext}
\getchunk{defun npNull}
\getchunk{defun npParenclosed}
\getchunk{defun npParenthesize}
\getchunk{defun npParenthesized}
\getchunk{defun npParse}
\getchunk{defun npPDefinition}
\getchunk{defun npPileBracketed}
\getchunk{defun npPileDefinitionlist}
\getchunk{defun npPileExit}
\getchunk{defun npPower}
\getchunk{defun npPP}
\getchunk{defun npPPf}
\getchunk{defun npPPff}
\getchunk{defun npPPg}
\getchunk{defun npPrefixColon}
\getchunk{defun npPretend}
\getchunk{defun npPrimary}
\getchunk{defun npPrimary1}
\getchunk{defun npPrimary2}
\getchunk{defun npProcessSynonym}
\getchunk{defun npProduct}
\getchunk{defun npPushId}
\getchunk{defun npRelation}
\getchunk{defun npRemainder}
\getchunk{defun npQualDef}
\getchunk{defun npQualified}
\getchunk{defun npQualifiedDefinition}
\getchunk{defun npQualType}
\getchunk{defun npQualTypelist}
\getchunk{defun npQuiver}
\getchunk{defun npRecoverTrap}
\getchunk{defun npRestore}
\getchunk{defun npRestrict}
\getchunk{defun npReturn}
\getchunk{defun npRightAssoc}
\getchunk{defun npRule}
\getchunk{defun npSCategory}
\getchunk{defun npSDefaultItem}
\getchunk{defun npSegment}
\getchunk{defun npSelector}
\getchunk{defun npSemiBackSet}
\getchunk{defun npSemiListing}
\getchunk{defun npSigDecl}
\getchunk{defun npSigItem}
\getchunk{defun npSigItemlist}
\getchunk{defun npSignature}
\getchunk{defun npSignatureDefines}
(defun optionUserLevelError)
(defun orderBySlotNumber)
(defun originsInOrder)
(defun parseAndEval)
(defun parseAndEval1)
(defun parseAndInterpret)
(defun parseFromString)
(defun parseNoMacroFromString)
(defun parseSystemCmd)
(defun parseWord)
(defun pathname)
(defun pathnameDirectory)
(defun pathnameName)
(defun pathnameType)
(defun pathnameTypeId)
(defun patternVarsOf)
(defun patternVarsOf1)
(defun permuteToOrder)
(defun pcounters)
(defun pfAbSynOp)
(defun pfAbSynOp?)
(defun pfAdd)
(defun pfAnd)
(defun pfAnd?)
(defun pfApplication)
(defun pfApplication?)
(defun pfApplication2Sex)
(defun pfAssign)
(defun pfAssign?)
(defun pfAttribute)
(defun pfBrace)
(defun pfBraceBar)
(defun pfBracket)
(defun pfBracketBar)
(defun pfBreak)
(defun pfBreak?)
(defun pfCharPosn)
(defun pfCheckArg)
(defun pfCheckMacroOut)
(defun pfCheckId)
(defun pfCheckItOut)
(defun pfCoerceTo)
(defun pfCoerceTo?)
(defun pfCollect)
(defun pfCollect?)
(defun pfCollect1?)
(defun pfCollectArgTran)
(defun pfCollectVariable1)
(defun pfCollect2Sex)
CHAPTER 45. THE INTERPRETER

chunk{defun pfCopyWithPos}
chunk{defun pfDefinition}
chunk{defun pfDefinition?}
chunk{defun pfDefinition2Sex}
chunk{defun pfDo}
chunk{defun pfDo?}
chunk{defun pfDocument}
chunk{defun pfEnSequence}
chunk{defun pfExit}
chunk{defun pfExit?}
chunk{defun pfExport}
chunk{defun pfExpression}
chunk{defun pfFileName}
chunk{defun pfFix}
chunk{defun pfFlattenApp}
chunk{defun pfFree}
chunk{defun pfFree?}
chunk{defun pfForin}
chunk{defun pfForin?}
chunk{defun pfFromDom}
chunk{defun pfFromDom?}
chunk{defun pfGlobalLinePosn}
chunk{defun pfHide}
chunk{defun pfId}
chunk{defun pfId?}
chunk{defun pfIdPos}
chunk{defun pfIdSymbol}
chunk{defun pfIf}
chunk{defun pfIf?}
chunk{defun pfIfThenOnly}
chunk{defun pfImport}
chunk{defun pfInline}
chunk{defun pfInfApplication}
chunk{defun pfIterate}
chunk{defun pfIterate?}
chunk{defun pfLam}
chunk{defun pfLamBase}
chunk{defun pfLambda}
chunk{defun pfLambdaTran}
chunk{defun pfLambda2Sex}
chunk{defun pfLeaf}
chunk{defun pfLeaf?}
chunk{defun pfLeafPosition}
chunk{defun pfLeafToken}
chunk{defun pfLhsRule2Sex}
chunk{defun pfLinePosn}
chunk{defun pfListOf}
chunk{defun pfLiteralClass}
chunk{defun pfLiteralString}
\getchunk{defun pfLiteral2Sex}
\getchunk{defun pfLocal}
\getchunk{defun pfLocal?}
\getchunk{defun pfLoop}
\getchunk{defun pfLoop1}
\getchunk{defun pfLoop?}
\getchunk{defun pfLp}
\getchunk{defun pfMacro}
\getchunk{defun pfMacro?}
\getchunk{defun pfMapParts}
\getchunk{defun pfMLambda}
\getchunk{defun pfMLambda?}
\getchunk{defun pfname}
\getchunk{defun pfNoPosition}
\getchunk{defun pfNoPosition?}
\getchunk{defun pfNot?}
\getchunk{defun pfNothing}
\getchunk{defun pfNothing?}
\getchunk{defun pfNovalue}
\getchunk{defun pfNovalue?}
\getchunk{defun pfOp2Sex}
\getchunk{defun pfOr}
\getchunk{defun pfOr?}
\getchunk{defun pfParen}
\getchunk{defun pfPretend}
\getchunk{defun pfPretend?}
\getchunk{defun pfPushBody}
\getchunk{defun pfPushMacroBody}
\getchunk{defun pfQualType}
\getchunk{defun pfRestrict}
\getchunk{defun pfRestrict?}
\getchunk{defun pfRetractTo}
\getchunk{defun pfReturn}
\getchunk{defun pfReturn?}
\getchunk{defun pfReturnNoName}
\getchunk{defun pfReturnType}
\getchunk{defun pfRhsRule2Sex}
\getchunk{defun pfRule}
\getchunk{defun pfRule?}
\getchunk{defun pfRule2Sex}
\getchunk{defun pfSequence}
\getchunk{defun pfSequence?}
\getchunk{defun pfSequenceToList}
\getchunk{defun pfSequence2Sex}
\getchunk{defun pfSequence2Sex0}
\getchunk{defun pfSexpr}
\getchunk{defun pfSexpr,strip}
\getchunk{defun pfSourcePosition}
\getchunk{defun pfSourceStok}
\getchunk{defun pfSpread}
\defun{pfSuch}
\defun{pfSuchthat}
\defun{pfSuchthat?}
\defun{pfSuchThat2Sex}
\defun{pfSymbol}
\defun{pfSymbol?}
\defun{pfSymbolSymbol}
\defun{pfTagged}
\defun{pfTagged?}
\defun{pfTaggedToTyped}
\defun{pfTaggedToTyped1}
\defun{pfTransformArg}
\defun{pfTuple}
\defun{pfTupleListOf}
\defun{pfTweakIf}
\defun{pfTyped}
\defun{pfTyped?}
\defun{pfTyping}
\defun{pfTuple?}
\defun{pfUnSequence}
\defun{pfWDec}
\defun{pfWDeclare}
\defun{pfWhere}
\defun{pfWhere?}
\defun{pfWhile}
\defun{pfWhile?}
\defun{pfWith}
\defun{pfWrong}
\defun{pfWrong?}
\defun{pf0ApplicationArgs}
\defun{pf0DefinitionLhsItems}
\defun{pf0FlattenSyntacticTuple}
\defun{pf0ForinLhs}
\defun{pf0FreeItems}
\defun{pf0LambdaArgs}
\defun{pf0LocalItems}
\defun{pf0LoopIterators}
\defun{pf0LoopLambdas}
\defun{pf0SequenceArgs}
\defun{pf0TupleParts}
\defun{pf0WhereContext}
\defun{pf2Sex}
\defun{pf2Sex1}
\defun{phMacro}
\defun{phParse}
\defun{phInterpret}
\defun{phIntReportMsgs}
\defun{pileCforest}
\defun{pileColumn}
\getchunk{defun pileCtree}
\getchunk{defun pileForest}
\getchunk{defun pileForest1}
\getchunk{defun pileForestes}
\getchunk{defun pilePlusComment}
\getchunk{defun pilePlusComments}
\getchunk{defun pileTree}
\getchunk{defun poFileName}
\getchunk{defun poGlobalLinePosn}
\getchunk{defun poLinePosn}
\getchunk{defun poPosImmediate?}
\getchunk{defun porigin}
\getchunk{defun posend}
\getchunk{defun posPointers}
\getchunk{defun ppos}
\getchunk{defun pquit}
\getchunk{defun pquitSpad2Cmd}
\getchunk{defun previousInterpreterFrame}
\getchunk{defun printLabelledList}
\getchunk{defun printStatisticsSummary}
\getchunk{defun printStorage}
\getchunk{defun printSynonyms}
\getchunk{defun printTypeAndTime}
\getchunk{defun probeName}
\getchunk{defun processChPosesForOneLine}
\getchunk{defun processInteractive}
\getchunk{defun processInteractive1}
\getchunk{defun processKeyedError}
\getchunk{defun processMsgList}
\getchunk{defun protectedEVAL}
\getchunk{defun processSynonymLine}
\getchunk{defun processSynonymLine,removeKeyFromLine}
\getchunk{defun processSynonyms}
\getchunk{defun prTraceNames}
\getchunk{defun prTraceNames,fn}
\getchunk{defun pSearch}
\getchunk{defun spacers}
\getchunk{defun ptimers}
\getchunk{defun put}
\getchunk{defun putFTText}
\getchunk{defun punctuation?}
\getchunk{defun putDatabaseStuff}
\getchunk{defun putHist}
\getchunk{defun pvarCondList1}
\getchunk{defun pvarCondList}
\getchunk{defun pvarPredTran}
\getchunk{defun pvarsOfPattern}

\getchunk{defun queryClients}
\getchunk{defun queueUpErrors}
defun quit
\getchunk{defun quitSpad2Cmd}
defun quoteString
\getchunk{defun rassocSub}
defun rdefinstream
defun rdefoutstream
defun read
\getchunk{defun /read}
defun readHiFi
defun readline
\getchunk{defun readSpadProfileIfThere}
defun readSpad2Cmd
\getchunk{defun reassembleTowerIntoType}
defun recordAndPrint
defun recordFrame
defun recordNewValue
defun recordNewValue0
defun recordOldValue
defun recordOldValue0
defun reduceAlistForDomain
\getchunk{defun redundant}
defun regress command
defun regress
\getchunk{defun remFile}
defun removeOption
\getchunk{defun remover}
defun removeTracedMapSigs
defun removeUndoLines
\getchunk{defun renamePatternVariables1}
defun renamePatternVariables
defun replaceFile
\getchunk{defun replacePercentByDollar,fn}
defun replacePercentByDollar
\getchunk{defun replaceSharps}
defun reportAO
\getchunk{defun reportInstantiations}
defun reportOperations
\getchunk{defun reportOpsFromLisplib}
defun reportOpsFromLisplib0
\getchunk{defun reportOpsFromLisplib1}
defun reportOpsFromUnitDirectly
\getchunk{defun reportOpsFromUnitDirectly0}
defun reportOpsFromUnitDirectly1
\getchunk{defun reportSpadTrace}
defun reportUndo
\getchunk{defun reportWhatOptions}
defun reroot
\getchunk{defun resetCounters}
defun resethashTables
\getchunk{defun resetInCoreHist}
\getchunk{defun resetSpacers}
\getchunk{defun resetTimers}
\getchunk{defun resetWorkspaceVariables}
\getchunk{defun restart}
\getchunk{defun restart0}
\getchunk{defun restoreHistory}
\getchunk{defun retract}
\getchunk{defun retractByFunction}
\getchunk{defun retractUnderDomain}
\getchunk{defun retract2Specialization}
\getchunk{defun rread}
\getchunk{defun ruleLhsTran}
\getchunk{defun rulePredicateTran}
\getchunk{defun runspad}
\getchunk{defun rwrite}

\getchunk{defun safeWritify}
\getchunk{defun sameMsg?}
\getchunk{defun satisfiesRegularExpressions}
\getchunk{defun saveHistory}
\getchunk{defun saveMapSig}
\getchunk{defun savesystem}
\getchunk{defun saveDependentsHashTable}
\getchunk{defun saveUsersHashTable}
\getchunk{defun sayAllCacheCounts}
\getchunk{defun sayBrightly1}
\getchunk{defun sayCacheCount}
\getchunk{defun sayExample}
\getchunk{defun sayKeyedMsg}
\getchunk{defun sayKeyedMsgLocal}
\getchunk{defun sayMSG}
\getchunk{defun sayMSG2File}
\getchunk{defun sayShowWarning}
\getchunk{defun scanCheckRadix}
\getchunk{defun scanComment}
\getchunk{defun scanDictCons}
\getchunk{defun scanError}
\getchunk{defun scanEsc}
\getchunk{defun scanEscape}
\getchunk{defun scanExponent}
\getchunk{defun scanIgnoreLine}
\getchunk{defun scanInsert}
\getchunk{defun scanKeyTr}
\getchunk{defun scanNegComment}
\getchunk{defun scanNumber}
\getchunk{defun ScanOrPairVec}
\getchunk{defun ScanOrPairVec,ScanOrInner}
\getchunk{defun scanPossFloat}
\getchunk{defun scanPunct}
\getchunk{defun scanPunCons}
\getchunk{defun scanS}
\getchunk{defun scanSpace}
\getchunk{defun scanString}
\getchunk{defun scanKeyTableCons}
\getchunk{defun scanToken}
\getchunk{defun scanTransform}
\getchunk{defun scanW}
\getchunk{defun scanWord}
\getchunk{defun search}
\getchunk{defun searchCurrentEnv}
\getchunk{defun searchTailEnv}
\getchunk{defun segmentKeyedMsg}
\getchunk{defun selectOption}
\getchunk{defun selectOptionLC}
\getchunk{defun separatePiles}
\getchunk{defun serverReadLine}
\getchunk{defun set}
\getchunk{defun set1}
\getchunk{defun setdatabase}
\getchunk{defun setExposed}
\getchunk{defun setExposeAdd}
\getchunk{defun setExposeAddConstr}
\getchunk{defun setExposeAddGroup}
\getchunk{defun setExposeDrop}
\getchunk{defun setExposeDropConstr}
\getchunk{defun setExposeDropGroup}
\getchunk{defun setFortDir}
\getchunk{defun setFortPers}
\getchunk{defun setFortTmpDir}
\getchunk{defun setFunctionsCache}
\getchunk{defun setHistoryCore}
\getchunk{defun setInputLibrary}
\getchunk{defun setIOindex}
\getchunk{defun setLinkerArgs}
\getchunk{defun setMsgCatlessAttr}
\getchunk{defun setMsgForcedAttr}
\getchunk{defun setMsgForcedAttrList}
\getchunk{defun setMsgUnforcedAttr}
\getchunk{defun setMsgUnforcedAttrList}
\getchunk{defun setNagHost}
\getchunk{defun setOutputAlgebra}
\getchunk{defun setOutputCharacters}
\getchunk{defun setOutputFormula}
\getchunk{defun setOutputFortran}
\getchunk{defun setOutputLibrary}
\getchunk{defun setOutputHtml}
\getchunk{defun setOutputMathml}
\getchunk{defun setOutputOpenMath}
\getchunk{defun setOutputTex}
\getchunk{defun setStreamsCalculate}
\getchunk{defun setUpDefault}
\getchunk{defun shortenForPrinting}
\getchunk{defun show}
\getchunk{defun showdatabase}
\getchunk{defun showHistory}
\getchunk{defun showInOut}
\getchunk{defun showInput}
\getchunk{defun showSpad2Cmd}
\getchunk{defun shut}
\getchunk{defun size}
\getchunk{defun SkipEnd?}
\getchunk{defun SkipPart?}
\getchunk{defun Skipping?}
\getchunk{defun spad}
\getchunk{defun spadClosure?}
\getchunk{defun SpadInterpretStream}
\getchunk{defun spad-long-error}
\getchunk{defun spadReply}
\getchunk{defun spadReply,printName}
\getchunk{defun spadrread}
\getchunk{defun spadrwrite}
\getchunk{defun spadrwrite0}
\getchunk{defun spad-save}
\getchunk{defun spad-short-error}
\getchunk{defun spadStartUpMgs}
\getchunk{defun spad-syntax-error}
\getchunk{defun spadTrace}
\getchunk{defun spadTraceAlias}
\getchunk{defun spadTrace,g}
\getchunk{defun spadTrace,isTraceable}
\getchunk{defun spadUntrace}
\getchunk{defun spad2BootCoerce}
\getchunk{defun specialChar}
\getchunk{defun spleI}
\getchunk{defun spleI1}
\getchunk{defun split}
\getchunk{defun splitIntoOptionBlocks}
\getchunk{defun stackTraceOptionError}
\getchunk{defun startp}
\getchunk{defun startsNegComment?}
\getchunk{defun statisticsInitialization}
\getchunk{defun streamChop}
\getchunk{defun size}
\getchunk{defun stringList2String}
\getchunk{defun stringMatches?}
\getchunk{defun StringToDir}
\getchunk{defun stripUnionTags}
\getchunk{defun strpos}
\getchunk{defun strposl}
\getchunk{defun stupidIsSpadFunction}
\getchunk{defun subMatch}
\getchunk{defun substFromAlist}
\getchunk{defun substringMatch}
\getchunk{defun subTypes}
\getchunk{defun summary}
\getchunk{defun syGeneralErrorHere}
\getchunk{defun syIgnoredFromTo}
\getchunk{defun synonym}
\getchunk{defun synonymsForUserLevel}
\getchunk{defun synonymSpad2Cmd}
\getchunk{defun sySpecificErrorAtToken}
\getchunk{defun sySpecificErrorHere}
\getchunk{defun systemCommand}

\getchunk{defun ?t}
\getchunk{defun tabbing}
\getchunk{defun templateParts}
\getchunk{defun tangle}
\getchunk{defun terminateSystemCommand}
\getchunk{defun tersyscommand}
\getchunk{defun testnumberp}
\getchunk{defun testpassed}
\getchunk{defun thisPosIsEqual}
\getchunk{defun thisPosIsLess}
\getchunk{defun throwEvalTypeMsg}
\getchunk{defun toFile?}
\getchunk{defun tokConstruct}
\getchunk{defun token-stack-show}
\getchunk{defun tokPosn}
\getchunk{defun tokTran}
\getchunk{defun tokType}
\getchunk{defun topLevelInterpEval}
\getchunk{defun trace}
\getchunk{defun trace1}
\getchunk{defun traceDomainConstructor}
\getchunk{defun traceDomainLocalDps}
\getchunk{defun tracelet}
\getchunk{defun traceOptionError}
\getchunk{defun /tracereply}
\getchunk{defun traceReply}
\getchunk{defun traceSpad2Cmd}
\getchunk{defun translateTrueFalse2YesNo}
\getchunk{defun translateYesNo2TrueFalse}
\getchunk{defun translateYesNoToTrueFalse}
\getchunk{defun transOnlyOption}
\getchunk{defun transTraceItems}
\getchunk{defun typeCheckInputAreas}

defun unAbbreviateKeyword

defun undo

defun undoChanges

defun undoCount

defun undoFromFile

defun undoInCore

defun undoLocalModemapHack

defun undoSingleStep

defun undoSteps

defun unescapeStringsInForm

defun unifyStruct

defun unifyStructVar

defun unparseInputForm

defun untrace

defun untraceDomainConstructor

defun untraceDomainConstructor,keepTraced?

defun untraceDomainLocalOps

defun untraceMapSubNames

defun unwriteable?

defun upcase

defun updateCurrentInterpreterFrame

defun updateDatabase

defun updateFromCurrentInterpreterFrame

defun updateHist

defun updateInCoreHist

defun updateSourceFiles

defun userLevelErrorMessage

defun validateOutputDirectory

defun valueArgsEqual?

defun vec2list

defun voidValue

defun what

defun whatCommands

defun whatConstructors

defun whatSpad2Cmd

defun whatSpad2Cmd,fixpat

defun whichCat

defun with

defun workfiles

defun workfilesSpad2Cmd

defun wrap

defun write-browsedb

defun write-categorydb

defun writeHiFi

defun writeHistModesAndValues

defun writeInputLines

defun write-interpdb

\getchunk{defun write-operationdb}
\getchunk{defun write-warmdata}
\getchunk{defun writify}
\getchunk{defun writifyComplain}
\getchunk{defun writify,writifyInner}
\getchunk{defun xlCannotRead}
\getchunk{defun xlCmdBug}
\getchunk{defun xlConActive}
\getchunk{defun xlConsole}
\getchunk{defun xlConStill}
\getchunk{defun xlFileCycle}
\getchunk{defun xlIfBug}
\getchunk{defun xlIfSyntax}
\getchunk{defun xlMsg}
\getchunk{defun xlNoSuchFile}
\getchunk{defun xlOK}
\getchunk{defun xlOK1}
\getchunk{defun xlPrematureEOF}
\getchunk{defun xlPrematureFin}
\getchunk{defun xlSay}
\getchunk{defun xlSkip}
\getchunk{defun xlSkippingFin}
\getchunk{defun xSearch}
\getchunk{defun yesanswer}
\getchunk{defun ySearch}
\getchunk{postvars}
Chapter 46

The Global Variables

46.1 Star Global Variables

<table>
<thead>
<tr>
<th>NAME</th>
<th>SET</th>
<th>USE</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>eof</em></td>
<td>ncTopLevel</td>
<td></td>
</tr>
<tr>
<td>features*</td>
<td>restart</td>
<td></td>
</tr>
<tr>
<td>package*</td>
<td>restart</td>
<td></td>
</tr>
<tr>
<td>standard-input*</td>
<td>ncIntLoop</td>
<td></td>
</tr>
<tr>
<td>standard-output*</td>
<td>ncIntLoop</td>
<td></td>
</tr>
<tr>
<td>top-level-hook*</td>
<td>set-restart-hook</td>
<td></td>
</tr>
</tbody>
</table>

*eof*

The *eof* variable is set to NIL in ncTopLevel.

*features*

The *features* variable from common lisp is tested for the presence of the :unix keyword. Apparently this controls the use of Saturn, a previous Axiom frontend. The Saturn frontend was never released as open source and so this test and the associated variables are probably not used.

*package*

The *package* variable, from common lisp, is set in restart to the BOOT package where the interpreter lives.
*standard-input*

The *standard-input* common lisp variable is used to set the curinstream variable in ncIntLoop.

This variable is an argument to serverReadLine in the intloopReadConsole function.

*standard-output*

The *standard-output* common lisp variable is used to set the curoutstream variable in ncIntLoop.

*top-level-hook*

The *top-level-hook* common lisp variable contains the name of a function to invoke when an image is started. In our case it is called restart. This is the entry point to the Axiom interpreter.
46.1. STAR GLOBAL VARIABLES
## Chapter 46. The Global Variables

### 46.2 Dollar Global Variables

<table>
<thead>
<tr>
<th>NAME</th>
<th>SET</th>
<th>USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$boot</td>
<td>ncTopLevel</td>
<td>runspad</td>
</tr>
<tr>
<td>coerceFailure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>curinstream</td>
<td>ncntLoop</td>
<td></td>
</tr>
<tr>
<td>curoutstream</td>
<td>ncntLoop</td>
<td></td>
</tr>
<tr>
<td>$currentLine</td>
<td>restart</td>
<td>removeUndoLines</td>
</tr>
<tr>
<td>$dalymode</td>
<td></td>
<td>intloopReadConsole</td>
</tr>
<tr>
<td>$displayStartMsgs</td>
<td>ncntoplevel</td>
<td>restart</td>
</tr>
<tr>
<td>$e</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$erMsgToss</td>
<td>SpadInterpretStream</td>
<td></td>
</tr>
<tr>
<td>$frameRecord</td>
<td>initvars</td>
<td>clearFrame</td>
</tr>
<tr>
<td></td>
<td></td>
<td>undoSteps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>recordFrame</td>
</tr>
<tr>
<td>$HiFiAccess</td>
<td>initHist</td>
<td>historySpad2Cmd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>setHistoryCore</td>
</tr>
<tr>
<td>$HistList</td>
<td>initHist</td>
<td></td>
</tr>
<tr>
<td>$HistListAct</td>
<td>initHist</td>
<td></td>
</tr>
<tr>
<td>$HistListLen</td>
<td>initHistList</td>
<td></td>
</tr>
<tr>
<td>$HistRecord</td>
<td>initHistList</td>
<td></td>
</tr>
<tr>
<td>$historyDirectory</td>
<td>initHist</td>
<td>makeHistFileName</td>
</tr>
<tr>
<td></td>
<td></td>
<td>makeHistFileName</td>
</tr>
<tr>
<td>$historyFileType</td>
<td>initvars</td>
<td>histInputFileName</td>
</tr>
<tr>
<td>$InteractiveFrame</td>
<td>restart</td>
<td>ncntoplevel</td>
</tr>
<tr>
<td></td>
<td>undo</td>
<td>recordFrame</td>
</tr>
<tr>
<td></td>
<td>undoSteps</td>
<td>recordFrame</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reportUndo</td>
</tr>
<tr>
<td>$internalHistoryTable</td>
<td>initvars</td>
<td></td>
</tr>
<tr>
<td>$interpreterFrameName</td>
<td>initializeInterpreterFrameRing</td>
<td></td>
</tr>
<tr>
<td>$interpreterFrameRing</td>
<td>initializeInterpreterFrameRing</td>
<td></td>
</tr>
<tr>
<td>$intRestart</td>
<td>ncntoplevel</td>
<td></td>
</tr>
<tr>
<td>$intTopLevel</td>
<td>initloop</td>
<td></td>
</tr>
<tr>
<td>$IOindex</td>
<td>restart</td>
<td>historySpad2Cmd</td>
</tr>
<tr>
<td></td>
<td>removeUndoLines</td>
<td>undoCount</td>
</tr>
<tr>
<td>$genValue</td>
<td>bookvol5</td>
<td>i-toplev</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i-analy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i-spec1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i-spec2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i-map</td>
</tr>
<tr>
<td>$lastPos</td>
<td>SpadInterpretStream</td>
<td></td>
</tr>
<tr>
<td>$libQuiet</td>
<td>SpadInterpretStream</td>
<td></td>
</tr>
<tr>
<td>$msgDatabaseName</td>
<td>reroot *</td>
<td></td>
</tr>
<tr>
<td>$ncMsgList</td>
<td>SpadInterpretStream</td>
<td></td>
</tr>
<tr>
<td>$newcompErrorCount</td>
<td>SpadInterpretStream</td>
<td></td>
</tr>
<tr>
<td>$newspad</td>
<td>ncntoplevel</td>
<td></td>
</tr>
<tr>
<td>$nopos</td>
<td></td>
<td>SpadInterpretStream</td>
</tr>
<tr>
<td>$okToExecuteMachineCode</td>
<td>SpadInterpretStream</td>
<td></td>
</tr>
<tr>
<td>$oldHistoryFileName</td>
<td>initvars</td>
<td>oldHistFileName</td>
</tr>
<tr>
<td>$options</td>
<td></td>
<td>historySpad2Cmd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>undo</td>
</tr>
</tbody>
</table>
46.2. DOLLAR GLOBAL VARIABLES

$boot

The $boot variable is set to NIL in ncTopLevel.

coerceFailure

The coerceFailure symbol is a catch tag used in runspad to catch an exit from ncTopLevel.

$currentLine

The $currentLine line is set to NIL in restart. It is used in removeUndoLines in the undo mechanism.

$displayStartMsgs

The $displayStartMsgs variable is used in restart but is not set so this is likely a bug.

$erMsgToss

The $erMsgToss variable is set to NIL in SpadInterpretStream.

$frameRecord

$frameRecord = [delta1, delta2, ...] where delta(i) contains changes in the “backwards” direction. Each delta(i) has the form ((var . proplist)...) where proplist denotes an ordinary proplist. For example, an entry of the form ((x (value) (mode (Integer))))... indicates that to undo 1 step, x’s value is cleared and its mode should be set to (Integer).

A delta(i) of the form (systemCommand . delta) is a special delta indicating changes due to system commands executed between the last command and the current command. By recording these deltas separately, it is possible to undo to either BEFORE or AFTER the command. These special delta(i)s are given ONLY when a system command is given which alters the environment.

recordFrame('system) is called before a command is executed, and recordFrame('normal) is called after (see processInteractive1). If no changes are found for former, no special entry is given.

This is part of the undo mechanism.

$intRestart

The $intRestart variable is used in intloop but has no value. This is probably a bug. While the variable’s value is unchanged the system will continually reenter the SpadInterpretStream
$\text{intTopLevel}$

The $\text{intTopLevel}$ is a catch tag. Throwing to this tags which is caught in the intloop will restart the SpadInterpretStream function.

$\text{IOindex}$

The $\text{IOindex}$ index variable is set to 1 in restart. This variable is used in the historySpad2Cmd function in the history mechanism. It is set in the removeUndoLines function in the undo mechanism.

This is used in the undo mechanism in function undoCount to compute the number of undos. You can’t undo more actions then have already happened.

$\text{lastPos}$

The $\text{lastPos}$ variable is set in SpadInterpretStream to the value of the $\text{nopos}$ variable. Since $\text{nopos}$ appears to have no value this is likely a bug.

$\text{libQuiet}$

The $\text{libQuiet}$ variable is set to the third argument of the SpadInterpretStream function. This is passed from intloop with the value of T. This variable appears to be intended to control the printing of library loading messages which would need to be suppressed if input was coming from a file.

$\text{msgDatabaseName}$

The $\text{msgDatabaseName}$ is set to NIL in reroot.

$\text{ncMsgList}$

The $\text{ncMsgList}$ is set to NIL in SpadInterpretStream.

$\text{newcompErrorCount}$

The $\text{newcompErrorCount}$ is set to 0 in SpadInterpretStream.
46.2. DOLLAR GLOBAL VARIABLES

$nopos

The $nopos variable is used in SpadInterpretStream but does not appear to have a value and is likely a bug.

$oldHistoryFileName

The $oldHistoryFileName is set at load time by a call to initvars to a value of “last”. It is part of the history mechanism. It is used in the function oldHistFileName and restoreHistory.

$okToExecuteMachineCode

The $okToExecuteMachineCode is set to T in SpadInterpretStream.

$options

The $options variable is tested by the history function. If it is NIL then output the message

You have not used the correct syntax for the history command.
Issue )help history for more information.

The $options variable is tested in the historySpad2Cmd function. It appears to record the options that were given to a spad command on the input line. The function selectOptionLC appears to take a list off options to scan.

This variable is not yet set and is probably a bug.

$previousBindings

The $previousBindings is a copy of the CAAR $InteractiveFrame. This is used to compute the delta(i)s stored in $frameRecord. This is part of the undo mechanism.

$reportundo

The $reportundo variable is used in diffAlist. It was not normally bound but has been set to T in initvars. If the variable is set to T then we call reportUndo.

It is part of the undo mechanism.

$spad

The $spad variable is set to T in ncTopLevel.
$SpadServer

If an open server is not requested then this variable to T. It has no value before this time (and is thus a bug).

$SpadServerName

The $SpadServerName is passed to the openServer function, if the function exists.

$systemCommandFunction

The $systemCommandFunction is set in SpadInterpretStream to point to the function Inter-
pExecuteSpadSystemCommand.

top_level

The top_level symbol is a catch tag used in runspad to catch an exit from ncTopLevel.

$quitTag

The $quitTag is used as a variable in a catch block. It appears that it can be thrown somewhere below ncTopLevel.

$useInternalHistoryTable

The $useInternalHistoryTable variable is set at load time by a call to initvars to a value of NIL. It is part of the history mechanism.
Chapter 47

Signatures

237  set-restart-hook : Void → 'restart
253  intloopReadConsole : (String Integer) → Throw
259  intloopPrefix? : String → Union(String,NIL)
261  intloopProcessString : (String,StepNo) → StepNo
262  next : (Function,Delay) → Delay
262  next1 : Delay → ParsePair
263  incString : String → Function
266  setCurrentLine : String → List(String)
266  mkprompt : Void → String
268  serverReadLine : Stream → String
288  intloopProcess : (StepNo,Boolean,Delay) → StepNo
298  incRenumber : Delay → Delay
298  incZip : (Function,Delay,Delay) → Delay
299  incZip1 : Delay → ParsePair
299  incIgen : Integer → Delay
301  incLude : (Int,List(String),Int,List(String),List(Int)) → Delay
326  incCommand? : String → Boolean
329  Delay : (Function,List(Any)) → Delay
555  StreamNull : Delay → Union(T,NIL)
1334  getHtMacroItem : String → Values (String NonNegativeInteger)
Chapter 48

Bibliography

**Abstract:** One principle of structured programming is that a program should be separated into meaningful independent subprograms, which are then combined so that the relation of the parts to the whole can be clearly established. This paper describes several alternative ways to compose programs. The main method used is to permit the programmer to denote by an expression the sequence of values taken on by a variable. The sequence is represented by a function called a stream, which is a functional analog of a coroutine. The conventional while and for loops of structured programming may be composed by a technique of stream processing (analogous to list processing), which results in more structured programs than the originals. This technique makes it possible to structure a program in a natural way into its logically separate parts, which can then be considered independently.


Mike Dewar. Openmath: An overview.
Chapter 49

Index
Index

*ThisIsARunningSystem*, 891
defvar, 891

*all-tests-ran*, 882
used by regress, 882
defvar, 882

*allOperations*, 1146
used by allOperations, 1179
used by localnrlib, 1161
defvar, 1146

*allconstructors*, 1146
used by allConstructors, 1178
used by browseopen, 1151
used by interpopen, 1150
used by localnrlib, 1161
used by make-databases, 1165
used by resethashtables, 1146
defvar, 1146

*ancestors-hash*
used by write-interpdb, 1173

defvar, 1146

*build-version*
used by axiomVersion, 748
used by spadStartUpMsgs, 241

defvar, 1145

*category-stream*, 1145
used by categoryopen, 1152
used by getdatabase, 1156
defvar, 1145

*category-stream-stamp*, 1145
used by categoryopen, 1152
used by getdatabase, 1156
defvar, 1145

*defaultdomain-list*, 1142
used by getdatabase, 1156
defvar, 1142

*eof*, 246
used by ncTopLevel, 248
used by serverReadLine, 269
defvar, 246

*hasCategory-hash*, 1143
used by categoryopen, 1152
used by getdatabase, 1156
used by write-categorydb, 1176
defvar, 1143

*hascategory-hash*
used by getdatabase, 1156
used by resethashtables, 1147

defvar, 1147

*index-filename*
used by localdatabase, 1160

defvar, 1147

*monitor-domains*, 1313
used by monitor-readinterp, 1326
used by monitor-report, 1327
used by monitor-spadfile, 1328
defvar, 1313

*monitor-nrlibs*, 1313

1624
INDEX

used by monitor-dirname, 1324
defvar, 1313
*monitor-table*, 1314
used by monitor-add, 1316
used by monitor-apropos, 1329
used by monitor-checkpoint, 1321
used by monitor-decr, 1318
used by monitor-delete, 1316
used by monitor-disable, 1317
used by monitor-enable, 1316
used by monitor-end, 1315
used by monitor-incr, 1318
used by monitor-info, 1319
used by monitor-inittable, 1314
used by monitor-nrlib, 1325
used by monitor-percent, 1329
used by monitor-reset, 1317
used by monitor-results, 1315
used by monitor-tested, 1320
used by monitor-untested, 1319
defvar, 1314
*ok*, 884
used by startp, 887
used by testpassed, 885
defvar, 884
*operation-hash*, 1143
used by addoperations, 1153
used by allOperations, 1179
used by getdatabase, 1156
used by make-databases, 1165
used by operationopen, 1153
used by resethashstables, 1147
used by write-operationdb, 1177
defvar, 1143
*operation-stream*, 1144
used by getdatabase, 1156
used by operationopen, 1153
used by resethashstables, 1146
defvar, 1144
*operation-stream-stamp*, 1145
used by operationopen, 1153
used by resethashstables, 1146
defvar, 1145
*print-package*
used by monitor-checkpoint, 1321
*print-pretty*
used by write-browsedb, 1175
used by write-categorydb, 1176
used by write-interpdb, 1173
*read-place-holder*, 1227
used by placep, 1227
defvar, 1227
*sourcefiles*, 1164
used by make-databases, 1165
used by resethashstables, 1146
used by write-browsedb, 1175
defvar, 1164
*standard-output*
used by init-boot/spad-reader, 1114
*whitespace*, 246
used by assertCond, 321
defvar, 246
*yearweek*
used by axiomVersion, 748
used by spadStartUpMsgs, 241
/D,1
called by compileBoot, 1071
/breakcondition
used by break, 87
/countlist
used by pcounters, 1053
used by resetCounters, 1051
used by resetWorkspaceVariables, 894
/editfile, 781
used by /read, 877
used by editSpad2Cmd, 804
used by readSpad2Cmd, 875
used by readSpadProfileIfThere, 1107
used by resetWorkspaceVariables, 894
defvar, 781
/pretty
used by resetWorkspaceVariables, 894
/read, 876
called by readSpad2Cmd, 875
calls /rf[9], 876
calls /rq[9], 876
uses /editfile, 877
defun, 876
/RF[9]
called by /read, 876
/RQ[9]
called by /read, 876
INDEX

/sourcefiles
  usedby resetWorkspaceVariables, 894
/spacelist
  usedby pspacers, 1052
  usedby resetSpacers, 1051
  usedby resetWorkspaceVariables, 894
/timerlist
  usedby ptimers, 1052
  usedby resetTimers, 1051
  usedby resetWorkspaceVariables, 894
/trace,0
  calledby trace1, 49
/tracenames
  usedby /tracerreply, 77
  usedby ?.t, 84
  usedby getBpiNameITracedMap, 75
  usedby getMapSubNames, 65
  usedby prTraceNames, 80
  usedby spadReply, 1069
  usedby spadTrace, 68
  usedby spadUntrace, 78
  usedby traceReply, 81
  usedby untraceDomainConstructor, 73
  usedby untrace, 60
/tracerreply, 77
  calledby untraceDomainConstructor,keepTraceIfVar, 73
  calledby untrace, 59
/untrace,2
  calledby untraceMapSubNames, 1059
?order
  calledby insertAlist, 913
?t, 84
  calledby trace1, 48
  calls bright, 84
  calls devaluate, 84
  calls get, 84
  calls isDomainOrPackage, 84
  calls isDomain, 84
  calls isgenvar, 84
  calls qcar, 84
  calls qcdr, 84
  calls rassocSub, 84
  calls reportSpadTrace, 84
  calls sayBrightly, 84
  calls sayMSG, 84
  calls take, 84
  uses /tracenames, 84
  uses $InteractiveFrame, 84
  uses $mapSubNameAlist, 84
defun, 84
$FINDFILE
  calledby $editSpad2Cmd, 804
$erase
  calledby $addNewInterpreterFrame, 18
  calledby $closeInterpreterFrame, 22
$fcopy
  calledby $restoreHistory, 839
$filename
  calledby $editSpad2Cmd, 804
$erase
  calledby $addNewInterpreterFrame, 18
  calledby $closeInterpreterFrame, 22

$AnonymousFunction, 640
  usedby coerceInt1, 676
defvar, 640
$Any, 640
  usedby coerceInt0, 674
  usedby coerceInt1, 676
  usedby retract2Specialization, 704
defvar, 640
$BFtag, 640
used by bcTaylorSeries, 1367
used by coerceInt1, 676
used by coerceInteractive, 672
used by displayValue, 731
used by interpret2, 280
used by recordAndPrint, 282
defvar, 640
$EndOfOutput
  used by serverReadLine, 269
$EndServerSession, 268
    used by serverReadLine, 269
defvar, 268
$EndSession
    used by serverReadLine, 269
$Exit, 641
defvar, 641
$Expression, 641
defvar, 641
$Float, 642
defvar, 642
$FontTable, 643
defvar, 643
$FormalMapVariableList
  local ref domArg, 652
  local ref mkDomPvar, 663
  local ref replaceSharps, 1104
  used by dbConstructorDoc,hn, 1538
  used by dbGetDocTable,hn, 1541
  used by dbSubConform, 1544
  used by displayOperationsFromLisplib, 1027
  used by localnrlib, 1161
  used by reportOpsFromLisplib, 1025
$FunctionalExpression, 646
  local ref defaultTargetFE, 668
  defvar, 646
$HTCompanionWindowID, 277
  used by recordAndPrint, 282
defvar, 277
$HiFiAccess, 31, 937
  used by createCurrentInterpreterFrame, 14
  used by disableHist, 845
  used by emptyInterpreterFrame, 12
  used by historySpad2Cnd, 822
  used by initHist, 820
  used by putHist, 832
  used by restoreHistory, 839
used by saveHistory, 837
used by setHistoryCore, 826
used by undoFromFile, 836
used by undoInCore, 834
used by updateFromCurrentInterpreterFrame, 15
used by updateHist, 831
used by writeInputLines, 829
defvar, 31, 937
$HistListAct, 32
  used by changeHistListLen, 830
  used by createCurrentInterpreterFrame, 14
  used by emptyInterpreterFrame, 12
  used by initHistList, 821
  used by resetInCoreHist, 830
  used by updateFromCurrentInterpreterFrame, 15
  used by updateInCoreHist, 832
defvar, 32
$HistListLen, 32
  used by changeHistListLen, 830
  used by createCurrentInterpreterFrame, 14
  used by emptyInterpreterFrame, 12
  used by initHistList, 821
  used by resetInCoreHist, 830
  used by undoInCore, 834
  used by updateFromCurrentInterpreterFrame, 15
  used by updateInCoreHist, 832
defvar, 32
$HistList, 31
  used by changeHistListLen, 830
  used by createCurrentInterpreterFrame, 14
  used by emptyInterpreterFrame, 12
  used by initHistList, 821
  used by recordOldValue0, 834
  used by resetInCoreHist, 830
  used by undoChanges, 835
  used by undoInCore, 834
  used by updateFromCurrentInterpreterFrame, 15
  used by updateInCoreHist, 832
defvar, 31
$HistRecord, 32
  used by createCurrentInterpreterFrame, 14
  used by emptyInterpreterFrame, 12
used by initHistList, 821
used by recordNewValue0, 833
used by updateFromCurrentInterpreterFrame, 15
used by updateHist, 831
used by writeHiFi, 844
defvar, 32
$IIndex, 24
    used by createCurrentInterpreterFrame, 14
used by historySpad2Cmd, 822
used by mkprompt, 267
used by removeUndoLines, 41
used by resetWorkspaceVariables, 895
used by restart, 239
used by setHistoryCore, 826
used by setIOindex, 841
used by undoCount, 45
used by undoInCore, 834
used by undoSteps, 38
used by updateFromCurrentInterpreterFrame, 15
used by updateHist, 831
used by writeHiFi, 844
used by writeInPutLines, 829
defvar, 24
$InitialCommandSynonymAlist, 748
    used by resetWorkspaceVariables, 895
defvar, 748
$InitialModemapFrame, 130
    used by makeInitialModemapFrame, 261
defvar, 130
$Integer, 643
    local ref defaultTargetFE, 668
    local ref hasCateSpecialNew, 666
    local ref hasCateSpecial, 665
    local ref mkEvalable, 1076
    used by absolutelyCanCoerceByCheating, 702
used by coerceInt1, 676
used by coerceRetract, 709
used by retract2Specialization, 704
defvar, 643
$InteractiveFrame, 23
    local ref isDomainValuedVariable, 1105
used by ?, 84
used by augmentTraceNames, 65
used by clearCmdAll, 771
used by clearCmdParts, 773
used by createCurrentInterpreterFrame, 14
used by getAliasIfTracedMapParameter, 75
used by getBpiNameIfTracedMap, 75
used by getMapSig, 54
used by getMapSubNames, 65
used by getPreviousMapSubNames, 1057
used by getSubDomainPredicate, 701
used by getWorkspaceNames, 727
used by interpFunctionDepAlists, 737
used by isInterpOnlyMap, 1058
used by isUncompiledMap, 1058
used by ncTopLevel, 248
used by parseAndInterpret, 272
used by processInteractive1, 277
used by recordFrame, 1083
used by reportundo, 44
used by restart, 239
used by restoreHistory, 839
used by showSpad2Cmd, 1019
used by undoChanges, 835
used by undoFromFile, 836
used by undoInCore, 834
used by undoSteps, 38
used by undo, 36
used by updateFromCurrentInterpreterFrame, 15
used by writeHistModesAndValues, 845
defvar, 23
$InteractiveMode, 247
    local def unparseInputForm, 1238
    local ref loadLibNoUpdate, 1183
    local ref loadLib, 1182
    used by addBinding, 1109
    used by ncTopLevel, 248
    used by parseAndInterpret, 272
    used by readSpad2Cmd, 875
defvar, 247
$JoinOfCatDatabase
used by clearCmdCompletely, 770
$JoinOfDomDatabase
used by clearCmdCompletely, 770
$KillLispSystem
used by serverReadLine, 269
$LispCommand
  used by serverReadLine, 269
$MenuServer
  used by executeQuietCommand, 272
  used by serverReadLine, 269
$Mode, 643
defvar, 643
$NeedToSignalSessionManager, 268
  used by intloopSpadProcess, 289
  used by serverReadLine, 269
defvar, 268
$NegativeInteger, 643
defvar, 643
$NoValueMode
  used by coerceInteractive, 672
$NonNegativeInteger, 644
  used by coerceInt1, 676
  used by retract2Specialization, 704
defvar, 644
$NonNullStream, 853
  used by dewritify,dewritifyInner, 854
  used by writify,writifyInner, 850
defvar, 853
$NonPositiveInteger, 644
defvar, 644
$NonSmanSession
  used by serverReadLine, 269
$NullStream, 854
  used by dewritify,dewritifyInner, 854
  used by writify,writifyInner, 850
defvar, 854
$OutputForm, 642
  used by coerceByTable, 691
  used by coerceInt0, 674
  used by coerceInt1, 676
  used by coerceInteractive, 672
  used by coerceRetract, 709
  used by coerceSpadArgs2E, 63
  used by coerceSpadFunValue2E, 64
  used by coerceTraceArgs2E, 62
  used by coerceTraceFunValue2E, 64
defvar, 642
$PatternVariableList
  used by kDomainName, 1528
$PositiveInteger, 644
  used by coerceInt1, 676
  used by retract2Specialization, 704
defvar, 644
$Primitives
  used by conOpPage1, 1533
  used by dbShowConsDoc1, 1550
$PrintCompilerMessageIfTrue, 242
  used by spad, 242
defvar, 242
$ProcessInteractiveValue, 276
  used by kIsValidType, 1530
  used by processInteractive1, 277
  used by processInteractive, 275
  used by topLevelInterpEval, 1530
defvar, 276
$QueryClients
  used by queryClients, 777
$QuickLet
  used by breaklet, 86
  used by tracelet, 85
$QuietCommand, 271
  used by executeQuietCommand, 272
  used by pf2Sex1, 532
  used by pf2Sex, 529
  used by recordAndPrint, 282
defvar, 271
$QuietSpadCommand
  used by serverReadLine, 269
$QuotientField, 645
  local ref hasCateSpecial, 665
  used by coerceInt1, 676
  used by retract2Specialization, 704
defvar, 645
$RTspecialCharacters, 1124
  used by setOutputCharacters, 971
defvar, 1124
$RationalNumber, 644
  local ref defaultTargetFE, 668
  local ref hasCateSpecialNew, 666
defvar, 644
$SessionManager
  used by close, 777
  used by queryClients, 777
  used by serverReadLine, 269
$SingleFloat, 646
defvar, 646
$SingleInteger, 647
used by absolutelyCanCoerceByCheating, 702
used by coerceInt1, 676
used by coerceRetract, 709
defvar, 647
$SmallInteger, 646
defvar, 646
$SpadCommand
used by serverReadLine, 269
$SpadServerName, 133
used by restart, 239
defvar, 133
$SpadServer, 133
used by close, 777
used by restart, 239
used by serverReadLine, 269
used by spad-save, 1136
defvar, 133
$StreamFrame
used by processInteractive, 275
$StringCategory, 645
defvar, 645
$String, 645
used by coerceInt1, 676
defvar, 645
$Subst
local def ofCategory, 649
local ref hasCate, 664
local ref unifyStructVar, 659
$SwitchFrames
used by serverReadLine, 269
$Symbol, 645
local ref defaultTargetFE, 668
used by coerceInt1, 676
used by coerceRetract, 709
used by retract2Specialization, 704
defvar, 645
$ThrowAwayMode, 280
used by interpret2, 280
defvar, 280
$TriangleVariableList
local ref replaceSharps, 1104
used by dbShowConsDoc1, 1550
used by libConstructorSig, 1563
$UserAbbreviationsAlist
used by resetWorkspaceVariables, 895
$UserLevel, 1012
used by getDirectoryList, 1130
used by readSpad2Cmd, 875
used by satisfiesUserLevel, 723
used by set1, 1015
used by synonymsForUserLevel, 1038
used by userLevelErrorMessage, 723
used by whatCommands, 1094
defvar, 1012
$Void, 645
used by clearCmdSortedCaches, 769
used by coerceInt1, 676
used by recordAndPrint, 282
defvar, 645
$abbreviateTypes, 965
defvar, 965
$activePageList, 1403
defvar, 1403
$algebraFormat, 966
used by setOutputAlgebra, 967
defvar, 966
$algebraOutputFile, 966
used by setOutputAlgebra, 967
defvar, 966
$algebraOutputStream, 967
used by recordAndPrint, 282
used by sayMSG, 29
used by setOutputAlgebra, 967
defvar, 967
$analyzingMapList
used by processInteractive, 275
$ans
used by nplisp, 744
$args
used by dbConstructorDoc, 1538
$atLeastOneUnexposed
used by dbConsExposureMessage, 1549
used by kPage, 1497
$attrCats, 586
used by whichCat, 587
defvar, 586
$attributeDb
used by clearCmdCompletely, 770
$bcParseOnly, 1403
used by bcInputEquations, 1386
used by bcInputExplicitMatrix, 1339
used by `bcInputMatrixByFormula`, 1337
defvar, 1403

$boot, 248
  used by `ioclear`, 1118
  used by `iostat`, 1116
  used by `ncTopLevel`, 248
  used by `parseAndInterpret`, 272
  used by `resetWorkspaceVariables`, 894
defvar, 248

$cacheAlist, 911
  used by `countCache`, 912
  used by `sayAllCacheCounts`, 914
  used by `setFunctionsCache`, 912
defvar, 911

$cacheCount
  used by `countCache`, 913
  used by `sayAllCacheCounts`, 914
  used by `setFunctionsCache`, 912

$clearExcept, 767
  used by `clearSpad2Cmd`, 767
defvar, 767

$clearOptions, 766
  used by `clearCmdExcept`, 772
  used by `clearCmdParts`, 773
  used by `clearSpad2Cmd`, 767
defvar, 766

$coerceFailure
  used by `catchCoerceFailure`, 692
  used by `coerceByTable`, 691
  used by `coerceIntCommute`, 689
  used by `coerceRetract`, 709

$coerceIntByMapCounter
  used by `resetWorkspaceVariables`, 895

$collectOutput
  used by `printStatisticsSummary`, 283
  used by `printStorage`, 283
  used by `printTypeAndTime`, 284
  used by `recordAndPrint`, 282

$commentedOps
  used by `reportOpsFromUnitDirectly`, 1028

$comp ErrorMessageStack
  used by `processInteractive`, 275

$compilingLoop
  used by `processInteractive`, 275

$compilingMap, 274
  used by `coerceInteractive`, 672
  used by `processInteractive`, 275
defvar, 274

$conArgstrings
  used by `conPage`, 1503

$conformsAreDomains
  used by `kePage`, 1510

$conformsAreDomains
  used by `dbShowCons1`, 1547
  used by `dbSpecialDescription`, 1555
  used by `kPage`, 1497
  used by `kiPage`, 1509

$conform
  used by `conPageConEntry`, 1504
  used by `dbGetDocTable`, gn, 1542
  used by `dbGetDocTable`, ln, 1541
  used by `dbGetDocTable`, 1543

$conname
  used by `conPageConEntry`, 1504

$constructorList
  used by `make-databases`, 1165

$constructors, 1070
  used by `addTraceItem`, 84
  used by `traceReply`, 81
defvar, 1070

currentPage, 1403
  defvar, 1403

$current-directory, 129, 266
  used by `getDirectoryList`, 1130
  used by `reroot`, 265
  used by `restart`, 239
defvar, 129, 266

$current-line
  used by `iostat`, 1116
  used by `spad-short-error`, 1116

$currentCarrier
  used by `intloopSpadProcess`, 289

$currentFrameNum, 268
  used by `close`, 777
  used by `serverReadLine`, 269
defvar, 268

$currentLine
INDEX

used by ExecuteInterpSystemCommand, 256
  used by clearCmdAll, 771
used by getSystemCommandLine, 1038
used by intulpisp, 260
used by removeUndoLines, 41
used by restart, 239
used by setCurrentLine, 266
used by unAbbreviateKeyword, 741
used by updateHist, 831
used by writeHiFi, 844

$dalymode
  used by intloopReadConsole, 254
$dbDelimiters
  used by dbWordFrom, 1496
$declaredMode
  used by coerceConvertMmSelection, 684
  used by processInteractive, 275
$defaultFortVar
  used by processInteractive, 275
$defaultFortranType, 921
defvar, 921
$defaultFunctionTargets, 646
defvar, 646
$defaultPackageNamesHT
  used by kcPage, 1516
$defaultSpecialCharacters, 1121
defvar, 1121
$depTb
  local ref saveDependentsHashTable, 1168
$dependeeAlist
  used by displayProperties, sayFunctionDeps, 730
  used by displayProperties, 733
  used by interpFunctionDepAlists, 737
$dependeeClosureAlist
  used by resetWorkspaceVariables, 895
$dependentAlist
  used by displayProperties, sayFunctionDeps, 730
  used by displayProperties, 733
  used by interpFunctionDepAlists, 737
$describeOptions, 789
  used by describeSpad2Cmd, 790
defvar, 789
$directory-list, 130

used by getDirectoryList, 1130
used by reroot, 265
defvar, 130
$displayDroppedMap, 943
defvar, 943
$displayMsgNumber, 950
  used by sayKeyedMsgLocal, 28
defvar, 950
$displayOptions, 795
  used by displaySpad2Cmd, 796
defvar, 795
$displaySetValue, 952
  used by set1, 1015
defvar, 952
$displayStartMsgs, 953
  used by restart, 239
defvar, 953
$doNotAddEmptyModeIfTrue
  used by domainToGenvar, 1054
  used by reportOperations, 1022
  used by transTraceItem, 60
$docTableHash
  used by dbDocTable, 1539
$docTable
  used by dbAddDocTable, 1541
  used by dbDocTable, 1539
$doc
  used by conPageConEntry, 1504
$dollar
  used by retractByFunction, 710
$domPvar, 274
  local def hasAtt, 654
  local def hasCate1, 657
  local def hasCaty1, 662
  local def hasSig, 653
  local ref domArg2, 652
  local ref unifyStructVar, 659
$domainTraceNameAssoc
  used by genDomainTraceName, 59
$domains
  used by addTraceItem, 84
  used by traceReply, 81
$domain
used by dbSearchOrder, 1514
used by kTestPred, 1543
$echoLineStack
  used by resetWorkspaceVariables, 895
$envHashTable, 1109
  used by addBinding, 1109
defvar, 1109
$env, 247
  local ref isDomainValuedVariable, 1105
  used by getSubDomainPredicate, 701
  used by interpret, 278
  used by reportOperations, 1022
  used by resetWorkspaceVariables, 895
  used by showSpad2Cmd, 1019
defvar, 247
$erMsgToss
  used by SpadInterpretStream, 252
  used by initInPr, 589
  used by phIntReportMsgs, 291
  used by showMsgPos?, 580
$erase
  local ref saveDependentsHashTable, 1168
  local ref saveUsersHashTable, 1169
  used by deleteFile, 1193
  used by reportOpsFromLisplib1, 1024
  used by reportOpsFromUnitDirectly1, 1032
$evalDomain
  local ref evalDomain, 1075
$evalTimePrint
  used by showHistory, 825
$eval
  used by interpret1, 279
  used by interpret, 279
  used by reportOperations, 1022
$existingFiles
  used by clearCmdPartCompletely, 770
  used by resetWorkspaceVariables, 894
$expandSegments
  local def evaluateType, 1079
$exposed?
  used by conPageConEntry, 1504
$exposedOnlyIfTrue
  used by dbConsHeading, 1553
  used by dbShowCons1, 1547
  used by dbShowCons, 1545
$e, 247
  local ref isDomainValuedVariable, 1105
  used by clearCmdParts, 773
  used by coerceInt1, 676
  used by describeSpad2Cmd, 790
  used by displaySpad2Cmd, 796
  used by interpFunctionDepAlists, 737
  used by ncTopLevel, 248
  used by parseAndInterpret, 272
  used by processInteractive1, 277
  used by recordAndPrint, 282
  used by resetWorkspaceVariables, 895
  used by restoreHistory, 839
  used by retract2Specialization, 704
  used by showSpad2Cmd, 1019
  used by systemCommand, 720
  used by whatSpad2Cmd, 1091
defvar, 247
$filep
  used by setOutputAlgebra, 967
  used by setOutputFormula, 996
  used by setOutputFortran, 974
  used by setOutputHtml, 980
  used by setOutputMathml, 986
  used by setOutputOpenMath, 991
  used by setOutputTex, 1002
$findfile
  used by readSpad2Cmd, 875
$floatok
  used by lineoftoks, 335
  used by scanKeyTr, 343
  used by scanNumber, 355
  used by scanSpace, 352
  used by scanString, 353
  used by scanWord, 349
$fn
  used by SpadInterpretStream, 252
  used by toFile?, 585
$forceDatabaseUpdate
  local ref loadLib, 1182
  local ref updateDatabase, 1163
  used by localdatabase, 1160
$formatSigAsTex
  local def unparseInputForm, 1238
  used by setOutputFormula, 996
defvar, 994
defvar, 943
$globalExposureGroupAlist, 99
  local ref isExposedConstructor, 1026
  used by setExposeAddGroup, 89
  used by setExposeDropGroup, 95
defvar, 99
$hashOp0
  local ref basicLookup, 1186
$hashOp1
  local ref basicLookup, 1186
$hashOpApply
  local ref basicLookup, 1186
$hashOpSet
  local ref basicLookup, 1186
$hashSeg
  local ref basicLookup, 1186
$highlightAllowed, 946
defvar, 946
$historyDirectory, 819
  used by histInputFileName, 820
defvar, 819
$historyDisplayWidth, 935, 1475
defvar, 935, 1475
$historyFileType, 33, 818
defvar, 33, 818
$hope
  local def hasCate, 664
  local def ofCategory, 649
  local def unifyStructVar, 659
$htLineList, 1403
defvar, 1403
$htMacroTable, 1331
  used by buildHtMacroTable, 1333
defvar, 1331
$htMacroTable, 1331
  used by setOutputHtml, 980
defvar, 979
$htMacroTable, 1331
  used by setOutputHtml, 980
defvar, 979
$htMacroTable, 1331
  used by setOutputHtml, 980
defvar, 979
$htMacroTable, 1331
  used by setOutputHtml, 980
defvar, 979
$htMacroTable, 1331
  used by setOutputHtml, 980
defvar, 979
$htMacroTable, 1331
  used by setOutputHtml, 980
defvar, 979
$htMacroTable, 1331
  used by setOutputHtml, 980
defvar, 979
$instantCanCoerceCount
  used by processInteractive, 275
$instantCoerceCount
  used by processInteractive, 275
$instantMmCondCount
  used by processInteractive, 275
$imPrGuys, 580
defvar, 580
$imPrTagGuys, 589
  used by initImPr, 589
$imRetract
  used by processInteractive, 275
$inclAssertions
  used by SpadInterpretStream, 252
  used by assertCond, 321
  used by ifCond, 314
$includeUnexposed?
  used by getBrowseDatabase, 1241
$infovec
  used by dbAddChainDomain, 1544
  used by dbSearchOrder, 1514
$inputPromptType, 951
  used by mkprompt, 267
defvar, 951
$inputStream
  used by npFirstTok, 367
  used by npListAndRecover, 414
  used by npMoveTo, 415
  used by npNext, 370
  used by npParse, 365
  used by npRestore, 376
  used by npState, 436
$insideApplication
  used by pf2Sex, 529
  used by pfApplication2Sex, 536
  used by pfDefinition2Sex, 545
$insideCoerceInteractive
  used by coerceInteractive, 672
$insideRule
  used by pf2Sex1, 532
  used by pf2Sex, 529
  used by pfApplication2Sex, 536
  used by pfLhsRule2Sex, 549
  used by pfLiteral2Sex, 534
  used by pfOp2Sex, 538
  used by pfRhsRule2Sex, 549
$insideSEQ
  used by pf2Sex1, 532
  used by pf2Sex, 529
  used by pfSequence2Sex, 540
$instantCanCoerceCount
  used by processInteractive, 275
$instantCoerceCount
  used by processInteractive, 275
$instantMmCondCount
used by processInteractive, 275
$instantRecord, 274
used by processInteractive, 275
defvar, 274
$intCoerceFailure, 255
used by InterpExecuteSpadSystemCommand, 256
defvar, 255
$intRestart, 249
used by intloop, 250
defvar, 249
$intSpadReader, 255
used by InterpExecuteSpadSystemCommand, 256
defvar, 255
$intTopLevel, 249
used by intloop, 250
defvar, 249
$internalHistoryTable, 32
used by clearCmdAll, 771
used by createCurrentInterpreterFrame, 14
$interpreterTimedClasses
used by printStorage, 283
used by printTypeAndTime, 284
used by processInteractive, 275
$interpreterTimedNames
used by printStorage, 284
used by printTypeAndTime, 284
used by processInteractive, 275
$kind
used by conPageConEntry, 1504
$lambdaType, 902
defvar, 902
$lastLineInSEQ
used by processInteractive, 275
$lastPos
used by SpadInterpretStream, 252
used by getPosStL, 578
$lastUntraced
used by getMapSubNames, 65
used by trace1, 49
used by untraceMapSubNames, 1059
used by untrace, 60
$letAssoc
used by breaklet, 86
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>$macActive</td>
<td>$mac0ExpandBody, 453; $mac0MLambdaApply, 452; $macId, 455; macroExpanded, 450</td>
</tr>
<tr>
<td>$mapList</td>
<td>processInteractive, 275</td>
</tr>
<tr>
<td>$mapName</td>
<td>$mapList</td>
</tr>
<tr>
<td>$mapSubNameAlist</td>
<td>?t, 84; isSubForRedundantMapName, 1058; saveMapSig, 53; traceSpad2Cmd, 47; untraceMapSubNames, 1059; untrace, 60</td>
</tr>
<tr>
<td>$margin</td>
<td>msgText, 286; sayKeyedMsgLocal, 28; defvar, 983</td>
</tr>
<tr>
<td>$mathTraceList</td>
<td>coerceTraceArgs2E, 62; coerceTraceFunValue2E, 64; pfRule2Sex, 548; ruleLhsTran, 552; rulePredicateTran, 550</td>
</tr>
<tr>
<td>$mathmlFormat</td>
<td>setOutputMathml, 986; defvar, 985</td>
</tr>
<tr>
<td>$mathmlOutputFile</td>
<td>setOutputMathml, 986; defvar, 985</td>
</tr>
<tr>
<td>$mathmlOutputStream</td>
<td>setOutputMathml, 986; defvar, 985</td>
</tr>
<tr>
<td>$maximumFortranExpressionLength</td>
<td>defvar, 924</td>
</tr>
<tr>
<td>$minivectorCode</td>
<td>processInteractive, 275</td>
</tr>
<tr>
<td>$minivectorNames</td>
<td>processInteractive, 275</td>
</tr>
<tr>
<td>$minivector</td>
<td>processInteractive, 275</td>
</tr>
<tr>
<td>$mkTestFlag</td>
<td>recordAndPrint, 282; defvar, 281</td>
</tr>
<tr>
<td>$mkTestInputStack</td>
<td>updateHist, 831</td>
</tr>
<tr>
<td>$mkTestOutputType</td>
<td>recordAndPrint, 282; defvar, 281</td>
</tr>
<tr>
<td>$msgAlist</td>
<td>msgList</td>
</tr>
<tr>
<td>$msgDatabase</td>
<td>resetWorkspaceVariables, 895; defvar, 131; msgdbNoBlanksAfterGroup, 27; msgdbNoBlanksBeforeGroup, 27; msgdbPrims, 26; msgdbPunct, 27</td>
</tr>
<tr>
<td>$msgDatabaseName</td>
<td>resetWorkspaceVariables, 895; defvar, 131; msgdbNoBlanksAfterGroup, 27; msgdbNoBlanksBeforeGroup, 27; msgdbPrims, 26; msgdbPunct, 27</td>
</tr>
<tr>
<td>$msgdbNoBlanksAfterGroup</td>
<td>defvar, 27</td>
</tr>
<tr>
<td>$msgdbNoBlanksBeforeGroup</td>
<td>defvar, 27</td>
</tr>
<tr>
<td>$msgdbPrims</td>
<td>defvar, 26; fixObjectForPrinting, 728</td>
</tr>
<tr>
<td>$msgdbPunct</td>
<td>defvar, 27; fixObjectForPrinting, 728</td>
</tr>
<tr>
<td>$multiVarPredicateList</td>
<td>pfRule2Sex, 548; ruleLhsTran, 552; rulePredicateTran, 550</td>
</tr>
<tr>
<td>$nagEnforceDouble</td>
<td>defvar, 963; describeSetNagHost, 959; setNagHost, 959</td>
</tr>
<tr>
<td>$nagHost</td>
<td>defvar, 958; describeSetNagHost, 959; setNagHost, 959</td>
</tr>
<tr>
<td>$nagMessages</td>
<td>defvar, 949; describeSetNagHost, 959; setNagHost, 959</td>
</tr>
<tr>
<td>$ncMsgList</td>
<td>SpadInterpretStream, 252; intloopSpadProcess, 289; ncConversationPhase,wrapup, 293; ncConversationPhase, 293; processKeyedError, 575</td>
</tr>
<tr>
<td>$newConlist</td>
<td>library, 1159</td>
</tr>
<tr>
<td>$newcompErrorCount</td>
<td>SpadInterpretStream, 252; intloopSpadProcess, 289; ncConversationPhase,wrapup, 293; ncConversationPhase, 293; processKeyedError, 575</td>
</tr>
<tr>
<td>$newmapList</td>
<td>250</td>
</tr>
<tr>
<td>$newmapList</td>
<td>250</td>
</tr>
<tr>
<td>$newmapList</td>
<td>250</td>
</tr>
</tbody>
</table>
used by ncBug, 590
used by ncHardError, 574
used by ncSoftError, 574
defvar, 251
$newline, 1475
defvar, 1475
$newspad, 248
used by ncTopLevel, 248
defvar, 248
$noEvalTypeMsg, 1082
local ref throwEvalTypeMsg, 1082
used by kisValidType, 1530
used by topLevelInterpEval, 1530
defvar, 1082
$noParseCommands, 716
used by doSystemCommand, 718
defvar, 716
$noRepList
used by processMsgList, 591
used by redundant, 596
$noSubsumption, 1108
defvar, 1108
$nopos, 251
used by SpadInterpretStream, 252
used by makeLeaderMsg, 599
used by ncBug, 590
used by pfSourcePosition, 468
used by poNoPosition, 636
defvar, 251
$npPParg
used by npPPff, 435
$npTokToNames, 429
used by npId, 429
defvar, 429
$numberOfEquations, 1380
defvar, 1380
$numericFailure, 1242
defvar, 1242
$n
used by lineoftoks, 335
used by nextline, 336
used by scanCheckRadix, 357
used by scanComment, 339
used by scanError, 358
used by scanEscape, 358
used by scanEsc, 347
used by scanExponent, 350
used by scanNegComment, 340
used by scanNumber, 355
used by scanPossFloat, 344
used by scanPunct, 341
used by scanSpace, 352
used by scanString, 353
used by scanS, 354
used by scanToken, 338
used by scanW, 351
used by spleI1, 346
used by startsComment?, 339
used by startsNegComment?, 340
$okToExecuteMachineCode
used by SpadInterpretStream, 252
$oldBreakMode, 1242
defvar, 1242
$oldHistoryFileName, 818
used by oldHistFileName, 819
used by restoreHistory, 839
defvar, 818
$oldline, 721
used by commandErrorIfAmbiguous, 742
used by commandErrorMessage, 722
defvar, 721
$opSysName
used by spadStartUpMsgs, 241
$openMathFormat, 989
used by setOutputOpenMath, 991
defvar, 989
$openMathOutputStream
used by setOutputOpenMath, 991
defvar, 990
$openServerIfTrue, 131
used by restart, 239
used by spad-save, 1136
defvar, 131
$operationNameList
used by resetWorkspaceVariables, 895
$optionAlist, 1050
used by trace1, 49
defvar, 1050
$options
used by abbreviationsSpad2Cmd, 755
INDEX

used by clearSpad2Cmd, 767
used by close, 777
used by countCache, 912
used by frameSpad2Cm, 11
used by historySpad2Cm, 822
used by history, 821
used by library, 1159
used by readSpad2Cmd, 875
used by reportOpsFromUnitDirectly, 1028
used by restoreHistory, 839
used by setFunctionsCache, 911
used by showSpad2Cm, 1019
used by systemCommand, 720
used by trace1, 49
used by undo, 36
used by workfilesSpad2Cmd, 1099

$op
used by dbConstructorDoc:gn, 1539
used by dbConstructorDoc, 1539
used by dbGetDocTable, 1543
used by displayMacro, 726
used by displayType, 732
used by displayValue, 731
used by processInteractive, 275

$outputLibraryName
used by setOutputLibrary, 904

$outputLines
used by printTypeAndTime, 284

$outputList
used by processMsgList, 591
used by queueUpErrors, 594

$outputMode
used by recordAndPrint, 282

$packages
used by addTraceItem, 84
used by traceReply, 81

$pfMacros, 324
used by clearMacroTable, 772
used by clearParserMacro, 725
used by displayParserMacro, 736
used by getParserMacroNames, 725
used by mac0Define, 458
used by mac0GetName, 454
used by mac0Get, 456
used by mac0MLambdaApply, 452

used by macApplication, 451
used by macLambda,mac, 457
used by mac Lambda, 457
used by macWhere,mac, 457
used by macWhere, 456
defvar, 324

$plainRTspecialCharacters, 1124
used by setOutputCharacters, 971
defvar, 1124

$plainSpecialCharacters0, 1122
defvar, 1122

$plainSpecialCharacters1, 1122
defvar, 1122

$plainSpecialCharacters2, 1123
defvar, 1123

$plainSpecialCharacters3, 1123
defvar, 1123

$posActive
used by mac0ExpandBody, 453
used by mac0MLambdaApply, 452
used by macId, 455
used by macroExpanded, 450

$preLength, 577
used by getPreStl, 577
used by makeLeaderMsg, 599
used by makeMsgFromLine, 593
used by processChPosesForOneLine, 598
used by tabbing, 584
defvar, 577

$predicateList
used by pfRule2Sex, 548
used by pfSuchThat2Sex, 537
used by ruleLhsTran, 552

$predvec
used by dbSearchOrder, 1514
used by kTestSearchOrder, 1543

$prettyprint, 1011
defvar, 1011

$prevCarrier
used by intloopSpadProcess, 289

$previousBindings, 36
used by clearCmdAll, 771
used by clearFrame, 1088
used by recordFrame, 1083
defvar, 36

$primitiveHtCommands, 1331
used by buildHtMacroTable, 1333
defvar, 1331

$printAnyIfTrue, 939
used by recordAndPrint, 282
defvar, 939

$printFortranDecs, 921
defvar, 921

$printLoadMsgs, 940
local ref loadLibNoUpdate, 1183
local ref loadLib, 1182
used by restart, 239
defvar, 940

$printMsgsToFile, 944
used by sayKeyedMsgLocal, 28
defvar, 944

$printStatisticsSummaryIfTrue, 954
used by recordAndPrint, 282
defvar, 954

$printStoragedIfTrue
used by recordAndPrint, 282

$printTimeIfTrue, 955
used by printTypeAndTime, 284
used by recordAndPrint, 282
defvar, 955

$printTimeSum
used by showHistory, 825

$printTypeIfTrue, 956
used by printTypeAndTime, 284
used by recordAndPrint, 282
defvar, 956

$printVoidIfTrue, 957
used by recordAndPrint, 282
defvar, 957

$quadSymbol
local ref evaluateType1, 1081
used by reportOperations, 1022

$quitCommandType, 1005
used by pquitSpad2Cmd, 870
used by quitSpad2Cmd, 872
defvar, 1005

$quitTag, 243
used by runspad, 243
defvar, 243

$quotedOpList
used by pfOp2Sex, 538
used by pfRule2Sex, 548

$relative-directory-list, 132
used by reroot, 265
defvar, 132

$relative-library-directory-list, 132
used by reroot, 265
defvar, 132

$repGuys, 597
defvar, 597

$reportBottomUpFlag, 941
used by coerceConvertMmSelection, 684
used by retractByFunction, 710
defvar, 941

$reportCoerceIfTrue, 942
defvar, 942

$reportCompilation, 1010
defvar, 1010

$reportInstantiations, 947
used by processInteractive, 275
defvar, 947

$reportInterpOnly, 948
defvar, 948

$reportOptimization, 1010
defvar, 1010

$reportSpadTrace
used by spadTrace, 68

$reportSpadtrace, 1050
defvar, 1050

$reportundo, 36
defvar, 36

$runTestFlag, 281
used by recordAndPrint, 282
defvar, 281

$r
used by lineoftoks, 335
used by nextline, 336
used by scanEsc, 347

$sayBrightlyStream
used by reportOpsFromLisplib1, 1024
used by reportOpsFromUnitDirectly1, 1032

$seen
used by ScanOrPairVec,ScanOrInner, 858
used by ScanOrPairVec, 858
used by dewritify,dewritifyInner, 854
used by dewritify, 857
used by saveHistory, 837
used by writify,writifyInner, 850
INDEX

1643

used by writify, 852
$setOptionNames, 1013
used by set1, 1015
defvar, 1013
$setOptions
used by resetWorkspaceVariables, 895
used by set, 1014
$showOptions
used by reportOpsFromLisplib, 1025
used by reportOpsFromUnitDirectly, 1028
used by showSpad2Cmd, 1019
$.sig
used by dbConstructorDoc.hn, 1538
used by dbConstructorDoc, 1539
used by dbGetDocTable.hn, 1541
used by dbGetDocTable, 1543
$slamFlag
used by resetWorkspaceVariables, 895
$sockBufferLength, 268
used by serverReadLine, 269
defvar, 268
$solutionMethod, 1380
defvar, 1380
$sourceFiles
used by resetWorkspaceVariables, 894
used by updateSourceFiles, 806
used by workfilesSpad2Cmd, 1099
$spad-errors, 1113
used by init-boot/spad-reader, 1114
defvar, 1113
$spadroott, 133
local ref isSystemDirectory, 1183
used by DaaName, 1169
used by browseopen, 1151
used by categoryopen, 1152
used by getdatabase, 1156
used by initroot, 259
used by interpopen, 1150
used by make-absolute-filename, 260
used by operationopen, 1153
used by reroott, 265
used by write-browsedb, 1175
used by write-interpdb, 1173
defvar, 133
$spad
used by ioclear, 1118
used by iostat, 1116
used by ncTopLevel, 248
used by parseAndInterpret, 272
$specialCharacterAlist, 1125
used by setOutputCharacters, 971
used by specialChar, 1126
defvar, 1125
$specialCharacters, 1125
used by setOutputCharacters, 971
used by specialChar, 1126
defvar, 1125
$stack
used by npListAndRecover, 414
used by npListoFun, 445
used by npList, 379
used by npParse, 365
used by npPop1, 368
used by npPop2, 368
used by npPop3, 368
used by npPushId, 433
used by npPush, 367
used by npRestore, 376
used by npState, 436
used by npZeroOrMore, 401
$stepNo
used by intloopSpadProcess, 289
$stok
used by npConstTok, 409
used by npDDInfKey, 432
used by npDollar, 408
used by npEnclosed, 436
used by npEqKey, 369
used by npEqPeek, 376
used by npFirstTok, 367
used by npId, 429
used by npInfKey, 433
used by npInfixOperator, 384
used by npInfixOp, 385
used by npMissing, 376
used by npParenthesize, 439
used by npParse, 365
used by npPrefixColon, 386
used by npPushId, 433
used by npRecoverTrap, 415
used by npTrap, 437
used by sySpecificErrorHere, 417
$streamCount, 1006
    used by coerceSpadArgs2E, 63
    used by coerceSpadFunValue2E, 64
    used by describeSetStreamsCalculate, 1008
    used by setStreamsCalculate, 1007
defvar, 1006
$streamsShowAll, 1008
defvar, 1008
$superHash
    used by getSubDomainPredicate, 701
$syscommands, 716
    used by newHelpSpad2Cmd, 813
    used by systemCommand, 720
    used by unAbbreviateKeyword, 741
defvar, 716
$systemCommandFunction
    used by SpadInterpretStream, 252
    used by intloopProcess, 289
    used by nclloopCommand, 750
$systemCommands, 715
    used by synonymsForUserLevel, 1038
    used by systemCommand, 720
    used by unAbbreviateKeyword, 741
    used by whatCommands, 1094
defvar, 715
$systemType, 1379
defvar, 1379
$sz
    used by lineoftoks, 335
    used by nextline, 336
    used by scanComment, 339
    used by scanEsc, 347
    used by scanExponent, 350
    used by scanNegComment, 340
    used by scanNumber, 355
    used by scanPossFloat, 344
    used by scanS, 354
    used by scanW, 351
    used by splicI, 346
    used by startsComment?, 339
    used by startsNegComment?, 340
$testingErrorPrefix, 26
defvar, 26
$testingSystem, 955
defvar, 955
$texFormat, 1001
    used by setOutputTex, 1002
defvar, 1001
$texOutputFile, 1001
    used by setOutputTex, 1002
defvar, 1001
$texOutputStream
    used by printAsTeX, 285
    used by setOutputTex, 1002
$timeGlobalName
    used by processInteractive, 275
$timedNameStack
    used by interpretTopLevel, 278
$toWhereGuys, 585
defvar, 585
$tokenCommands, 747
    used by doSystemCommand, 718
defvar, 747
$topicHash
    used by write-warmdata, 1178
$traceErrorStack
    used by getTraceOptions, 52
    used by stackTraceOptionError, 1054
$traceNoisely, 1049
    used by reportSpadTrace, 76
    used by spadTrace, 68
    used by trace1, 49
defvar, 1049
$traceOptionList, 1050
    used by getTraceOption, 55
defvar, 1050
$tracedMapSignatures, 1050
    used by coerceTraceArgs2E, 62
    used by coerceTraceFunValue2E, 64
    used by removeTracedMapSigs, 61
    used by saveMapSig, 53
defvar, 1050
$traceModemap
    used by spadTrace, 67
$tracedSpadModemap
    used by coerceSpadArgs2E, 63
    used by coerceSpadFunValue2E, 64
$traceletFunctions
    used by breaklet, 86
    used by tracelet, 85
$traceletflag
    used by tracelet, 85
INDEX

$ttok
  used by npEqKey, 369
  used by npEqPeek, 376
  used by npFirstTok, 367
  used by npId, 429
  used by npInfKey, 433
  used by npInfixOp, 385
  used by np Parse, 365
  used by np PushId, 433
$underbar, 828
defvar, 828
$useBFasDefault
  used by float2Sex, 535
$useCoerceOrCroak
  used by coerceInt1, 676
$useConvertForCoercions
  used by coerceIntTest, 684
$useEditorForShowOutput, 999
  used by reportOpsFromLispLib0, 1023
  used by reportOpsFromUnitDirectly0, 1028
defvar, 999
$useFullScreenHelp, 936
  used by newHelpSpad2Cmd, 813
defvar, 936
$useInternalHistoryTable, 819
  used by clearCmdAll, 771
  used by initHist, 820
  used by readHiFi, 843
  used by restoreHistory, 839
  used by saveHistory, 837
  used by setHistoryCore, 826
  used by writeHiFi, 844
defvar, 819
$useIntrinsicFunctions, 923
defvar, 923
$usersTb
  local ref saveUsersHashTable, 1169
$variableNumberAlist
  used by clearCmdAll, 771
$whatOptions, 1090
  used by reportWhatsOptions, 1093
  used by whatSpad2Cmd, 1092
defvar, 1090
$whereCacheList
  used by processInteractive, 275
$which
  used by dbGetDocTable.hn, 1541
  used by dbGetDocTable, 1543
$writifyComplained
  used by writifyComplain, 848
  used by writify, 852
$xdatabase
  used by clearCmdCompletely, 770
abbreviate
  called by abbreviationsSpad2Cmd, 755
  called by displaySpad2Cmd, 796
calls getdatabase, 797
calls sayKeyedMsg, 797
defun, 797
abbreviation?
  called by abbreviationsSpad2Cmd, 755
  called by newHelpSpad2Cmd, 813
abbreviations, 753, 754
calls abbreviationsSpad2Cmd, 754
defun, 754
manpage, 753
abbreviationsSpad2Cmd, 754
called by abbreviations, 754
calls abbreviationsSpad2Cmd, 755
calls newHelpSpad2Cmd, 813
calls getdatabase, 797
calls exit, 755
calls helpSpad2Cmd, 755
calls listConstructorAbbreviations, 754
calls mkUserConstructorAbbreviation, 755
calls opOf, 755
calls qcar, 755
calls qcdr, 755
calls sayKeyedMsg, 755
calls selectOptionLC, 755
calls seq, 755
calls setdatabase, 755
calls size, 755
uses $options, 755
defun, 754
absolutelyCanCoerceByCheating, 702
called by absolutelyCanCoerceByCheating, 702
called by coerceInt1, 675
calls absolutelyCanCoerceByCheating, 702
uses $Integer, 702
defun, 702

acot, 1221
defun, 1221

acoth, 1224
defun, 1224

acsc, 1222
defun, 1222

acsch, 1223
defun, 1223

addassoc
calledby saveMapSig, 53
calledby trace1, 49

calls addBindingInteractive, 1109
calls getProplist, 1109
calls hput, 1109
uses $InteractiveMode, 1109
uses $envHashTable, 1109
defun, 1109

addBindingInteractive, 1113
calledby addBinding, 1109
calls assq, 1113
defun, 1113

addInputLibrary, 907
calledby setInputLibrary, 906
calls dropInputLibrary, 907
uses input-libraries, 907
defun, 907

addNewInterpreterFrame, 18
calledby frameSpad2Cmd, 11
calledby serverReadLine, 269
calls $erase, 18
calls boot-equal, 18
calls emptyInterpreterFrame, 18
callsframename, 18
calls histFileName, 18
calls initHistList, 18
calls throwKeyedMsg, 18
calls updateCurrentInterpreterFrame, 18
calls updateFromCurrentInterpreterFrame, 18

calls nequal, 702
defun, 702

calls deconstructT, 702

calls devaluate, 83
calls evalDomain, 696
calls spadcall, 696
uses $Boolean, 696
defun, 696

allConstructors, 1178
calledby make-databases, 1165
calledby write-browsedb, 1175
uses *allconstructors*, 1178
defun, 1178

allocate
calledby init-memory-configure, 258
allocate-contiguous-pages
calledby init-memory-configure, 258
allocate-relocatable-pages
calledby init-memory-configure, 258

allOperations, 1179
calledby apropos, 1097
uses *allOperations*, 1179
uses *operation-hash*, 1179
defun, 1179
INDEX

**alqlGetKindString**, 1236
calls dbPart, 1236
calls substring, 1236
defun, 1236

**alqlGetOrigin**, 1235
calls charPosition, 1235
calls dbPart, 1235
calls substring, 1235
defun, 1235

**alqlGetParams**, 1236
calls charPosition, 1236
calls dbPart, 1236
calls substring, 1236
defun, 1236

**alreadyOpened?**, 585
called by msgOutputter, 576
calls msgImPr?, 585
defun, 585

**ancestorsOf**
called by kcaPage, 1520
called by originsInOrder, 1540

**ancolsU16**, 1252
defmacro, 1252

**ancolsU32**, 1254
defmacro, 1254

**ancolsU8**, 1251
defmacro, 1251

**anrowsU16**, 1252
defmacro, 1252

**anrowsU32**, 1254
defmacro, 1254

**anrowsU8**, 1250
defmacro, 1250

**apropos**, 1097
called by whatSpad2Cmd, 1091
calls allOperations, 1097
calls downcase, 1097
calls exit, 1097
calls filterListOfStrings, 1097
calls isort, 1097
calls sayAsManyPerLineAsPossible, 1097
calls sayKeyedMsg, 1098
calls sayMessage, 1097
calls seq, 1097
defun, 1097

**aref2U16**, 1251
defmacro, 1251

**aref2U32**, 1253
defmacro, 1253

**aref2U8**, 1250
defmacro, 1250

**assoc**
called by breaklet, 86
called by clearCmdParts, 773
called by clearParserMacro, 725
called by diffAlist, 1085
called by domainDescendantsOf, 1507
called by getOption, 1068
called by koPageAux, 1537
called by readHiFi, 843
called by spadTrace, 67
called by spadUntrace, 78

**assocleft**
called by clearCmdParts, 773
called by dbShowCons1, 1547
called by isSubForRedundantMapName, 1059
called by originsInOrder, 1540

**assocright**
called by orderBySlotNumber, 1068
called by untraceMapSubNames, 1059

**assq**, 1200
called by addBindingInteractive, 1113
called by coerceIntTableOrFunction, 690
called by coerceIntTest, 684
called by diffAlist, 1085
called by fetchOutput, 842
called by hasSig, 653
called by recordNewValue0, 833
called by recordOldValue0, 833
called by searchCurrentEnv, 1110
called by searchTailEnv, 1111
called by showInOut, 842
called by specialChar, 1126
called by undoFromFile, 836
called by undoInCore, 834
called by undoSingleStep, 39
defmacro, 1200
asTupleAsList
called by coerceInt1, 675
augmentHasArgs, 1523
called by kcaPage1, 1521
called by kccPage, 1522
calls extractHasArgs, 1523
calls getConstructorForm, 1523
calls length, 1523
calls nreverse0, 1523
calls opOf, 1523
defun, 1523
augmentSub
called by hasCateSpecialNew, 666
called by hasCateSpecial, 665
called by hasCaty, 650
called by unifyStructVar, 659
augmentTraceNames, 65
called by traceSpad2Cmd, 47
calls get, 65
uses $InteractiveFrame, 65
defun, 65
AxiomServer
called by browse, 762
axiomVersion, 748
uses *build-version*, 748
uses *yearweek*, 748
defun, 748
AXSERV:axServer;IMV;2
called by browse, 762
basicLookup, 1185
calls HasCategory, 1186
calls error, 1186
calls hashCode?, 1186
calls hashString, 1186
calls hashType, 1186
calls spadcall, 1185
calls vecp, 1186
local ref $hashOp0, 1186
local ref $hashOp1, 1186
local ref $hashOpApply, 1186
local ref $hashOpSet, 1186
local ref $hashSeg, 1186
defun, 1185
basicLookupCheckDefaults, 1188
called by lookupInDomainVector, 1187
calls error, 1188
calls hashCode?, 1188
calls hashString, 1188
calls hashType, 1188
calls spadcall, 1188
calls vecp, 1188
local ref $lookupDefaults, 1188
defun, 1188
basicMatch?
called by stringMatches?, 1210
basicStringize, 1415
defun, 1415
bcAbbTable
called by dbShowCons1, 1547
bcComplexLimit, 1377
calls htInitPage, 1377
calls htMakePage, 1377
calls htShowPage, 1377
uses $EmptyMode, 1377
defun, 1377
bcComplexLimitGen, 1379
calls bcFinish, 1379
calls htpButtonValue, 1379
calls htpLabelInputString, 1379
calls strconc, 1379
defun, 1379
bcConPredTable
called by dbShowConditions, 1552
called by kePage, 1510
bcConTable
called by dbShowCons1, 1547
bcCreateVariableString, 1387
called by bcMakeEquations, 1388
called by bcMakeLinearEquations, 1388
defun, 1387

bcDefiniteIntegrate, 1344
calls htInitPage, 1344
calls htMakePage, 1344
calls htShowPage, 1344
uses $EmptyMode, 1344
defun, 1344

bcDefiniteIntegrateGen, 1345
calls bcGen, 1346
calls htpButtonValue, 1345
calls htpLabelInputString, 1345
calls strconc, 1345
defun, 1345

bcDifferentiate, 1349
calls htInitPage, 1349
calls htMakeDoneButton, 1349
calls htMakePage, 1349
calls htShowPage, 1349
uses $EmptyMode, 1349
defun, 1349

bcDifferentiateGen, 1350
calls bcError, 1350
calls bcGen, 1350
calls bcString2WordList, 1350
calls bcwords2liststring, 1350
calls htpLabelInputString, 1350
calls length, 1350
calls strconc, 1350
defun, 1350

bcDraw, 1351
calls bcHt, 1351
calls htInitPage, 1351
calls htShowPage, 1351
defun, 1351

bcDraw2Dfun, 1352
calls htInitPage, 1352
calls htMakePage, 1352
calls htShowPage, 1352
uses $EmptyMode, 1352
defun, 1352

bcDraw2DfunGen, 1353
calls bcDrawIt2, 1353
calls bcFinish, 1353
calls htpLabelInputString, 1353

calls strconc, 1353
defun, 1353

bcDraw2Dpar, 1353
calls htInitPage, 1353
calls htMakePage, 1354
calls htShowPage, 1354
uses $EmptyMode, 1354
defun, 1353

bcDraw2DparGen, 1355
calls bcDrawIt2, 1355
calls bcFinish, 1355
calls htpLabelInputString, 1355
calls strconc, 1355
defun, 1355

bcDraw2DSolve, 1355
calls htInitPage, 1355
calls htMakeDoneButton, 1355
calls htMakePage, 1355
calls htShowPage, 1356
uses $EmptyMode, 1356
defun, 1355

bcDraw2DSolveGen, 1357
calls bcFinish, 1357
calls htpLabelInputString, 1357
calls strconc, 1357
defun, 1357

bcDraw3Dfun, 1357
calls htInitPage, 1357
calls htMakePage, 1357
calls htShowPage, 1358
uses $EmptyMode, 1358
defun, 1357

bcDraw3DfunGen, 1359
calls bcDrawIt2, 1359
calls bcFinish, 1359
calls htpLabelInputString, 1359
calls strconc, 1359
defun, 1359

bcDraw3Dpar, 1360
calls htInitPage, 1360
calls htMakePage, 1360
calls htShowPage, 1360
uses $EmptyMode, 1360
defun, 1360

bcDraw3Dpar1, 1362
calls htInitPage, 1362
calls htMakeDoneButton, 1362
calls htMakePage, 1362
calls htShowPage, 1362
uses $EmptyMode, 1362
defun, 1362
bcDraw3Dpar1Gen, 1363
calls bcDrawIt2, 1363
calls bcFinish, 1363
calls htpLabelInputString, 1363
calls strconc, 1363
defun, 1363
bcDraw3DparGen, 1361
calls bcDrawIt2, 1361
calls bcFinish, 1361
calls htpLabelInputString, 1361
calls strconc, 1361
defun, 1361
bcDrawIt, 1401
calls strconc, 1401
defun, 1401
bcDrawIt2, 1342
calledby bcDraw2DfunGen, 1353
calledby bcDraw2DparGen, 1355
calledby bcDraw3DfunGen, 1359
calledby bcDraw3Dpar1Gen, 1363
calledby bcDraw3DparGen, 1361
calls bcMatrixGen, 1342
defun, 1342
bcError, 1401
calledby bcDifferentiateGen, 1350
calls sayBrightlyNT, 1401
calls sayBrightly, 1401
defun, 1401
bcErrorPage
calledby dbShowCons, 1545
bcFindString, 1398
defun, 1398
bcFinish, 1397
calledby bcComplexLimitGen, 1379
calledby bcDraw2DSolveGen, 1357
calledby bcDraw2DfunGen, 1353
calledby bcDraw2DparGen, 1355
calledby bcDraw3DfunGen, 1359
calledby bcDraw3Dpar1Gen, 1363
calledby bcDraw3DparGen, 1361
calledby bcLinearMatrixGen, 1394
calledby bcLinearSolveEqnsGen, 1396
calledby bcRealLimitGen1, 1377
calledby bcRealLimitGen, 1376
calledby bcSeriesExpansionGen, 1366
calledby bcSeriesGen, 1373
calledby bcSolveEquations, 1390
defun, 1397
bcGen, 1398, 1399
calledby bcDefiniteIntegrateGen, 1346
calledby bcDifferentiateGen, 1350
calledby bcGenExplicitMatrix, 1341
calledby bcIndefiniteIntegrateGen, 1344
calledby bcInputMatrixByFormulaGen, 1338
calledby bcLinearMatrixGen, 1394
calledby bcProductGen, 1348
calledby bcSumGen, 1347
calls htInitPage, 1398, 1399
calls htMakePage, 1398, 1399
calls htShowPage, 1398, 1399
calls strconc, 1398, 1399
defun, 1398, 1399
bcGenEquations, 1396
calledby bcLinearSolveEqnsGen, 1396
calledby bcSolveEquations, 1390
calls bcwords2liststring, 1397
calls strconc, 1396
defun, 1396
bcGenExplicitMatrix, 1341
calls bcGen, 1341
calls bcMatrixGen, 1341
calls htpInputAreaAlist, 1341
calls htpProperty, 1341
calls htpSetProperty, 1341
defun, 1341
bcHt, 1414
calledby bcDraw, 1351
calledby bcInputEquations, 1385
calledby bcInputExplicitMatrix, 1339
calledby bcvspace, 1400
calledby kdPageInfo, 1505
defun, 1414
bchtMakeButton, 1440
defun, 1440
bcIndefiniteIntegrate, 1343
calls htInitPage, 1343
calls htMakePage, 1343
calls htShowPage, 1343
uses $EmptyMode, 1343
defun, 1343
bcIndefiniteIntegrateGen, 1344
calls bcGen, 1344
calls htpLabelInputString, 1344
calls strconc, 1344
defun, 1344
bcInputEquations, 1385
calledby bcLinearSolveEqns, 1384
calledby bcSolveSingle, 1383
calledby bcSystemSolveEqns, 1384
calls bcHt, 1385
calls bcMakeEquations, 1385
calls bcMakeLinearEquations, 1385
calls bcMakeUnknowns, 1385
calls htInitPage, 1385
calls htMakeDoneButton, 1386
calls htMakePage, 1385
calls htProperty, 1385
calls htSay, 1385
calls htShowPage, 1386
calls htpSetPropertyList, 1385
calls htpSetProperty, 1385
calls objValUnwrap, 1385
calls parse-integer, 1385
calls strconc, 1385
uses $EmptyMode, 1386
uses $bcParseOnly, 1386
defun, 1385
bcInputEquationsEnd, 1389
calls systemError, 1389
defun, 1389
bcInputExplicitMatrix, 1339
calls bcHt, 1339
calls htInitPage, 1339
calls htMakeDoneButton, 1339
calls htMakePage, 1339
calls htShowPage, 1339
calls htpLabelInputString, 1339
calls htpPropertyList, 1339
calls htpSetProperty, 1339
calls objValUnwrap, 1339
calls parse-integer, 1339
calls strconc, 1339
uses $bcParseOnly, 1339
defun, 1339
bcInputExplicitMatrixByFormulaGen, 1338
calls bcGen, 1338
calls htpLabelInputString, 1338
calls htpPropertyList, 1338
calls htpSetProperty, 1338
defun, 1338
bcInputMatrixByFormula, 1337
calls htInitPage, 1337
calls htMakeDoneButton, 1337
calls htMakePage, 1337
calls htShowPage, 1337
calls htpLabelInputString, 1337
calls htpLabelSpadValue, 1337
calls htpSetProperty, 1337
calls objValUnwrap, 1337
calls parse-integer, 1337
uses $bcParseOnly, 1337
defun, 1337
bcInputMatrixByFormulaGen, 1338
calls bcGen, 1338
calls htpLabelInputString, 1338
calls htpPropertyList, 1338
bcInputSolveInfo, 1384
calls htInitPage, 1384
calls htMakePage, 1384
calls htShowPage, 1384
calls htpInputAreaList, 1384
calls htpPropertyList, 1384
calls htpSetProperty, 1384
bcIssueHt, 1414
defun, 1414
bcLaurentSeries, 1369
calls htInitPage, 1369
calls htMakePage, 1369
calls htShowPage, 1369
uses $EmptyMode, 1369
defun, 1369
bcLaurentSeriesGen, 1373
calls bcSeriesGen, 1373
defun, 1373
bcLimit, 1374
calls htInitPage, 1374
calls htMakePage, 1374
calls htShowPage, 1374
calls strconc, 1341
calls systemError, 1341
defun, 1341
bcMkFunction, 1397
called by bcLinearMatrixGen, 1394
defun, 1397
bcNameConTable
called by dbShowCons1, 1547
bcNotReady, 1401
called by bcSeriesByFormulaGen, 1369
calls htInitPage, 1401
calls htMakePage, 1401
calls htShowPage, 1401
defun, 1401
bcOptional, 1399
defun, 1399
bcProduct, 1348
calls htInitPage, 1348
calls htMakePage, 1348
calls htShowPage, 1348
uses $EmptyMode, 1348
defun, 1348
bcProductGen, 1348
calls bcGen, 1348
calls htpLabelInputString, 1348
defun, 1348
bcPuiseuxSeries, 1371
calls htInitPage, 1371
calls htMakePage, 1371
calls htShowPage, 1371
uses $EmptyMode, 1371
defun, 1371
bcPuiseuxSeriesGen, 1373
calls bcSeriesGen, 1373
defun, 1373
bcQueryInteger
called by finalExactRequest, 1395
called by linearFinalRequest, 1395
bcReadMatrix, 1336
called by bcLinearSolveMatrix, 1391
called by bcMatrix, 1336
calls htInitPage, 1336
calls htMakePage, 1336
calls htShowPage, 1336
calls htpSetProperty, 1336
defun, 1336
bcRealLimit, 1375
calls htInitPage, 1375
calls htMakePage, 1375
calls htShowPage, 1375
uses $EmptyMode, 1375
defun, 1375
bcRealLimitGen, 1376
called by bcFinish, 1376
called by htInitPage, 1376
called by htMakePage, 1376
called by htShowPage, 1376
called by htpButtonValue, 1376
called by htpLabelInputString, 1376
called by htpSetProperty, 1376
defun, 1376
bcRealLimitGen1, 1377
called by bcFinish, 1377
called by htpProperty, 1377
called by strconc, 1377
defun, 1377
bcSadFaces, 1422
defun, 1422
bcSeries, 1364
called by bcFinish, 1364
called by htInitPage, 1364
called by htMakePage, 1364
called by htShowPage, 1364
uses $EmptyMode, 1364
defun, 1364
bcSeriesByFormula, 1366
called by htInitPage, 1366
called by htMakePage, 1366
called by htShowPage, 1366
defun, 1366
bcSeriesByFormulaGen, 1369
called by bcNotReady, 1369
defun, 1369
bcSeriesExpansion, 1365
called by bcFinish, 1365
called by htInitPage, 1365
called by htMakeDoneButton, 1365
called by htMakePage, 1365
called by htShowPage, 1365
uses $EmptyMode, 1365
defun, 1365
bcSeriesExpansionGen, 1366
called by bcFinish, 1366
calls htpLabelInputString, 1366
calls strconc, 1366
defun, 1366
bcSeriesGen, 1373
called by bcLaurentSeriesGen, 1373
called by bcPuiseuxSeriesGen, 1373
called by bcTaylorSeriesGen, 1372
calls bcFinish, 1373
calls htpLabelInputString, 1373
calls strconc, 1373
defun, 1373
bcSolve, 1380
calls htpInitPage, 1380
calls htpMakePage, 1380
calls htpShowPage, 1380
uses $EmptyMode, 1380
defun, 1380
bcSolveEquations, 1390
called by bcSolveNumerically1, 1390
calls bcFinish, 1390
calls bcGenEquations, 1390
calls bcString2WordList, 1390
calls bcwords2liststring, 1390
calls htpButtonValue, 1390
calls htpLabelInputString, 1390
calls htpProperty, 1390
calls member, 1390
calls strconc, 1390
defun, 1390
bcSolveEquationsNumerically, 1389
calls htpInitPage, 1389
calls htpMakeDoneButton, 1389
calls htpMakePage, 1389
calls htpShowPage, 1389
calls htpPropertyList, 1389
defun, 1389
bcSolveNumerically1, 1390
calls bcSolveEquations, 1390
defun, 1390
bcSolveSingle, 1383
calls bcInputEquations, 1383
calls htpSetProperty, 1383
defun, 1383
bcString2HyString, 1398
defun, 1398
bcString2HyString2, 1398
defun, 1398
bcString2WordList, 1400
called by bcDifferentiateGen, 1350
called by bcLinearSolveEqnsGen, 1396
called by bcSolveEquations, 1390
defun, 1400
bcSum, 1346
calls htpInitPage, 1346
calls htpMakePage, 1346
calls htpShowPage, 1346
uses $EmptyMode, 1346
defun, 1346
bcSumGen, 1347
calls bcGen, 1347
calls htpLabelInputString, 1347
calls strconc, 1347
defun, 1347
bcSystemSolve, 1382
calls htpInitPage, 1382
calls htpMakeDoneButton, 1383
calls htpMakePage, 1382
calls htpShowPage, 1383
defun, 1382
bcSystemSolveEqns1, 1384
calls bcInputEquations, 1384
calls htpSetProperty, 1384
defun, 1384
bcTaylorSeries, 1367
calls htpInitPage, 1367
calls htpMakePage, 1367
calls htpShowPage, 1367
uses $EmptyMode, 1367
defun, 1367
bcTaylorSeriesGen, 1372
calls bcSeriesGen, 1372
defun, 1372
bcUnixTable, 1554
called by dbShowCons1, 1547
calls findfile, 1554
calls htBeginTable, 1554
calls htEndTable, 1555
calls htpMakePage, 1554
calls htpSay, 1554
calls namestring, 1554
defun, 1554
bcVectorGen, 1400
INDEX

1655

called by bcLinearMatrixGen, 1394
calls bcwords2liststring, 1400
defun, 1400
bcvspace, 1400
calls bcHt, 1400
defun, 1400
called by bcDifferentiateGen, 1350
called by bcGenEquations, 1397
called by bcLinearSolveEquationsGen, 1396
called by bcMatrixGen, 1341
called by bcSolveEquations, 1390
called by bcVectorGen, 1400
calls strconc, 1400
defun, 1400
beforeAfter, 1425
defun, 1425
bit-to-truth, 1232
defmacro, 1232
blankList
called by filterAndFormatConstructors, 1096
called by printLabelledList, 746
called by whatCommands, 1093
blocks, 608
syntax, 608
Boolean
called by hashable, 1246
BooleanEquality, 1206
defun, 1206
boot, 758
manpage, 758
boot-equal
called by addNewInterpreterFrame, 18
called by clearCmdParts, 773
called by displaySetOptionInformation, 896
called by fetchOutput, 842
called by findFrameInRing, 16
called by getMapSig, 54
called by importFromFrame, 19
called by setHistoryCore, 826
called by undoChanges, 835
called by untraceDomainConstructor,keepTraced?, 73
called by whatConstructors, 1097
called by writify,writifyInner, 849
boot-line-stack, 1107
used by init-boot/spad-reader, 1114
used by next-lines-clear, 1118
used by next-lines-show, 1117
defvar, 1107
bottomUp
called by coerceInt1, 675
called by evaluateType, 1078
called by interpret1, 279
bottomUp
called by evaluateType1, 1080
bpiname
called by breaklet, 86
called by hashable, 1247
called by mkEvalable, 1076
called by spadClosure?, 853
called by spadTrace,isTraceable, 66
called by spadTrace, 67
called by spadUntrace, 78
called by tracelet, 85
bpitrace
called by spadTrace, 67
bpuntrace
called by spadUntrace, 78
bracketString, 1449
defun, 1449
brCon
called by kcPage, 1515
break, 87
called by letPrint2, 1065
called by letPrint3, 1066
called by letPrint, 1062
calls MONITOR,EVALTRAN, 87
calls enable-backtrace, 87
calls interrupt, 87
calls sayBrightly, 87
uses /breakcondition, 87
defun, 87
breaklet, 86
calls assoc, 86
calls bpiname, 86
calls compileBoot, 86
calls delete, 86
calls gensymp, 86
calls lassoc, 86
calls setletprintflag, 86
calls stupidIsSpadFunction, 86
calls union, 86  
uses $QuickLet, 86  
uses $letAssoc, 86  
uses $traceletFunctions, 86  
defun, 86  
bright

calledby ?t, 84

calledby commandAmbiguityError, 724

calledby displayCondition, 736

calledby displayFrameNames, 13

calledby displayMacro, 726

calledby displayModemap, 738

calledby displayMode, 738

calledby displayProperties, sayFunctionDepths, 730

calledby displayProperties, 733

calledby displaySetOptionInformation, 896

calledby displaySetVariableSettings, 897

calledby letPrint2, 1065

calledby letPrint3, 1066

calledby letPrint, 1062

calledby pcounters, 1053

calledby pspacers, 1052

calledby ptimers, 1052

calledby reportOperations, 1021

calledby reportOpsFromLispLib, 1024

calledby reportOpsFromUnitDirectly, 1028

calledby sayCacheCount, 915

calledby set1, 1015

calledby setFortDir, 931

calledby setFortPers, 961

calledby setFortTempDir, 928

calledby setFunctionsCache, 911

calledby setOutputCharacters, 971

calledby setStreamsCalculate, 1007

calledby showHistory, 824

calledby spadUntrace, 78

brightprint, 1198

calledby saybrightly1, 1200

calls messageprint, 1198  
defun, 1198

brightprint-0, 1198

calledby saybrightly1, 1200

calls messageprint-1, 1198

defun, 1198

browse, 759

calls AXSERV;axServer;IMV;2, 762

calls AxiomServer, 762

calls loadLib, 762

calls set, 762

defun, 759

browsopen, 1151

calledby resethashtables, 1146

calledby restart0, 241

uses *alconstructors*, 1151

uses *browse-stream*, 1151

uses *browse-stream-stamp*, 1151

uses $spadroot, 1151

defun, 1151

browserAutoloadOnceTrigger

calledby make-databases, 1165

bubbleConstructor

calledby coerceIntTower, 682

buildHtMacroTable, 1333

calls concat, 1333

calls getHtMacroItem, 1333

gets environ, 1333

calls hput, 1333

calls sayBrightly, 1333

calls util.ht[7.1], 1333

uses $htMacroTable, 1333

uses $primitiveHtCommands, 1333

defun, 1333

buildLibdb

calledby make-databases, 1165

buildLibdbConEntry[9]
called by conPageConEntry, 1504

bumperrorcount

called by spad-syntax-error, 1115

buttonNames, 1429

defun, 1429

bvec-and, 1234

defun, 1234

bvec-concat, 1233

defun, 1233

bvec-copy, 1233

defun, 1233

bvec-elt, 1232

defmacro, 1232

bvec-equal, 1233

defun, 1233

bvec-greater, 1233
INDEX

defun, 1233
bvec-make-full, 1232
defun, 1232
bvec-nand, 1234
defun, 1234
bvec-nor, 1235
defun, 1235
bvec-not, 1235
defun, 1235
bvec-or, 1235
defun, 1235
bvec-setelt, 1232
defmacro, 1232
bvec-size, 1233
defmacro, 1233
bvec-xor, 1234
defun, 1234

callerName

calledby processKeyedError, 575
canCoerce

calledby unifyStructVar, 659
canCoerceFrom

calledby hasCateSpecial, 665
canFuncall?, 1197
calledby coerceRetract, 709
defun, 1197
capitalize

calledby dbConsHeading, 1553
calledby kPage, 1497
calledby kePage, 1509
calledby kiPage, 1509
calledby koPage, 1535
capitalize

calledby conOpPage1, 1533
Catch

calledby intloopSpadProcess, 289
CatchAsCan

calledby intloopSpadProcess, 289
catchCoerceFailure, 692
calls throwKeyedMsgCannotCoerceWith-
Value, 692
calls unwrap, 692
calls wrap, 692
uses $coerceFailure, 692
catches, 692
defun, 692
catches
catchCoerceFailure, 692
coerceByTable, 691
coerceInt1, 676
coerceIntByMap, 693
coerceIntCommute, 689
coerceRetract, 709
executeQuietCommand, 272
InterpExecuteSpadSystemCommand, 256
interpretTopLevel, 278
intloop, 250
intloopSpadProcess, 289
kDomainName, 1528
kisValidateType, 1530
letPrint2, 1064
letPrint3, 1066
monitor-file, 1319
monitor-readinterp, 1326
monitor-spadfile, 1328
npListAndRecover, 414
npParse, 365
runspad, 243
safeWritify, 849
ScanOrPairVec, 858
serverReadLine, 268, 269
categoryOpen?
calledby evaluateType1, 1080
calledby loadLib, 1182
calledby localnrlib, 1161
calledby make-databases, 1165
categoryopen, 1152
calledby resethashtables, 1146
calledby restart0, 241
uses *category-stream*, 1152
uses *category-stream-stamp*, 1152
uses *hasCategory-hash*, 1152
uses $spadroot, 1152
defun, 1152
cd, 764
manpage, 764
cdancols, 1208
defmacro, 1208
cdanrows, 1208
defmacro, 1208
cdarf2, 1207
defmacro, 1207
  cdelt, 1209
defmacro, 1209
cdlen, 1210
defmacro, 1210
cdsetaref2, 1207
defmacro, 1207
cdsetelt, 1209
defmacro, 1209
changeHistListLen, 830
called by historySpad2Cmd, 822
calls sayKeyedMsg, 830
uses $HistListAct, 830
uses $HistListLen, 830
uses $HistList, 830
defun, 830
changeToNamedInterpreterFrame, 17
called by serverReadLine, 269
calls findFrameInRing, 17
calls updateCurrentInterpreterFrame, 17
calls updateFromCurrentInterpreterFrame, 17
  uses $interpreterFrameRing, 17
defun, 17
char
  called by makeMsgFromLine, 593
charDigitVal, 859
called by gensymInt, 859
calls error, 859
defun, 859
charPosition
  called by alqlGetOrigin, 1235
called by alqlGetParams, 1236
called by conPageFastPath, 1504
called by removeUndoLines, 41
checkCondition, 1445
defun, 1445
checkFilter, 1493
calls trimString, 1493
defun, 1493
childrenOf
  called by kccPage, 1522
chkAllNonNegativeInteger, 1467
defun, 1467
chkDirectory, 1466
defun, 1466
chkNameList, 1465
defun, 1465
chkNonNegativeInteger, 1467
defun, 1467
chkOutputFileName, 1466
defun, 1466
chkPosInteger, 1466
defun, 1466
chkRange, 1467
defun, 1467
cleanline, 791
called by describeSpad2Cmd, 790
defun, 791
cleanupLine, 801
defun, 801
cleanupLine
  called by sayexample, 801
clear, 765, 767
calls clearSpad2Cmd, 767
defun, 767
manpage, 765
clearAllSlams
called by updateDatabase, 1163
clearClams
  called by clearCmdCompletely, 770
called by setExposeAddConstr, 93
called by setExposeAddGroup, 89
called by setExposeDropConstr, 96
called by setPoseDropGroup, 95
called by updateDatabase, 1163
clearCmdAll, 771
called by clearCmdCompletely, 770
called by clearFrame, 1088
called by clearSpad2Cmd, 767
calls clearCmdSortedCaches, 771
calls clearMacroTable, 771
calls deleteFile, 771
calls histFileName, 771
calls resetInCoreHist, 771
calls sayKeyedMsg, 771
calls untraceMapSubNames, 771
calls updateCurrentInterpreterFrame, 771
uses $InteractiveFrame, 771
uses $currentLine, 771
uses $frameMessages, 771
uses $frameRecord, 771
uses $\text{internalHistoryTable}, 771
uses $\text{interpreterFrameName}, 771
uses $\text{previousBindings}, 771
uses $\text{useInternalHistoryTable}, 771
uses $\text{variableNumberAlist}, 771
defun, 771
clearCmdCompletely, 770
calledby clearSpad2Cmd, 767
calls clearClams, 770
calls clearCmdAll, 770
calls clearConstructorCaches, 771
calls reclaim, 770
calls sayKeyedMsg, 770
uses $\text{CatOfCatDatabase}, 770
uses $\text{DomOfCatDatabase}, 770
uses $\text{JoinOfCatDatabase}, 770
uses $\text{JoinOfDomDatabase}, 770
uses $\text{attributeDb}, 770
uses $\text{existingFiles}, 770
uses $\text{functionTable}, 770
uses $\text{localExposureDataDefault}, 771
uses $\text{localExposureData}, 771
uses $\text{xdatabase}, 771
defun, 771
clearCmdExcept, 772
calledby clearSpad2Cmd, 767
calls clearCmdAll, 772
calls object2String, 772
calls stringPrefix?, 772
uses $\text{clearOptions}, 772
defun, 772
clearCmdParts, 773
calledby clearCmdExcept, 772
calledby clearSpad2Cmd, 767
calledby importFromFrame, 19
calls assocleft, 773
calls assoc, 773
calls boot-equal, 773
calls clearDependencies, 773
calls clearParserMacro, 773
calls deleteAssoc, 773
calls exit, 773
calls fixObjectForPrinting, 773
calls getInterpMacroNames, 773
calls getParserMacroNames, 773
calls get, 773
calls isMap, 773
calls member, 773
calls modes, 773
calls pname, 773
calls recordnewValue, 773
calls recordOldValue, 773
calls remdup, 773
calls sayKeyedMsg, 773
calls sayMessage, 773
calls selectOptionLC, 773
calls seq, 773
calls types, 773
calls untraceMapSubNames, 773
calls values, 773
uses $\text{InteractiveFrame}, 773
uses $\text{clearOptions}, 773
uses $\text{e}, 773
defun, 773
clearCmdSortedCaches, 769
calledby clearCmdAll, 771
calledby clearSpad2Cmd, 767
calledby restoreHistory, 839
calls compiledLookupCheck, 769
calls spadcall, 769
uses $\text{ConstructorCache}, 769
uses $\text{Void}, 769
uses $\text{lookupDefaults}, 769
defun, 769
clearConstructorCache

calledby loadLibNoUpdate, 1183
calledby loadLib, 1181
clearConstructorCaches

calledby clearCmdCompletely, 770
clearDependencies

calledby clearCmdParts, 773
clearDependentMaps

calledby coerceInteractive, 672
clearFrame, 1088
calls clearCmdAll, 1088
uses $\text{frameRecord}, 1088
uses $\text{previousBindings}, 1088
defun, 1088
clearMacroTable, 772
calledby clearCmdAll, 771
uses $\text{pfMacros}, 772
defun, 772
clearParserMacro, 725
  called by clearCmdParts, 773
calls assoc, 725
calls ifcdr, 725
calls remalist, 725
uses $pfMacros, 725
defun, 725
clearSpad2Cmd, 767
called by clear, 767
called by historySpad2Cmd, 822
called by restoreHistory, 839
calls clearCmdAll, 767
calls clearCmdCompletely, 767
calls clearCmdExcept, 767
calls clearCmdParts, 767
calls clearCmdSortedCaches, 767
calls sayKeyedMsg, 767
calls selectOptionLC, 767
calls updateCurrentInterpreterFrame, 767
uses $clearExcept, 767
uses $clearOptions, 767
defun, 767
clef, 610
syntax, 610
close, 776, 777
calls closeInterpreterFrame, 777
calls queryClients, 777
calls queryUserKeyedMsg, 777
calls selectOptionLC, 777
calls sockSendInt, 777
calls string2id-n, 777
calls throwKeyedMsg, 777
calls upcase, 777
uses $CloseClient, 777
uses $SessionManager, 777
uses $SpadServer, 777
uses $currentFrameNum, 777
uses $options, 777
defun, 777
manpage, 776
closeInterpreterFrame, 22
called by close, 777
called by frameSpad2Cmd, 11
calls $erase, 22
calls framerate, 22
calls makeHistFileName, 22
calls throwKeyedMsg, 22
calls updateFromCurrentInterpreterFrame, 22
uses $InterpreterFrameName, 22
uses $InterpreterFrameRing, 22
defun, 22
clrhash
  called by processInteractive, 274
cmpnote, 251
defun, 251
constructSubst
  called by hasSig, 653
coerceBranch2Union, 698
calls keyedSystemError, 698
calls mkObjWrap, 698
calls mkObj, 698
calls mkPredList, 698
calls objMode, 698
calls objVal, 698
calls orderUnionEntries, 698
calls position, 698
calls removeQuote, 698
calls stripUnionTags, 698
calls unwrap, 698
defun, 698
coerceByFunction, 680
called by coerceInt1, 675
called by coerceIntTableOrFunction, 691
defun, 680
coerceByTable, 691
called by coerceIntTableOrFunction, 690
calls isTotalCoerce, 691
calls isWrapped, 691
calls mkObjWrap, 691
calls mkObj, 691
calls mkq, 691
calls unwrap, 691
uses $OutputForm, 691
uses $coerceFailure, 691
catches, 691
defun, 691
coerceCommuteTest, 690
called by coerceIntCommute, 689
calls deconstructT, 690
calls isLegitimateMode, 690
calls underDomainOf, 690
defun, 690
coerceConvertMmSelection, 684
calledby coerceIntTest, 684
calls coerceConvertMmSelection;AL, 684
calls selectMms1, 684
uses $declaredMode, 684
uses $reportBottomUpFlag, 684
defun, 684
coerceConvertMmSelection;AL
calledby coerceConvertMmSelection, 684
calledby coerceSubDomain, 701
calledby getSubDomainPredicate, 701
defun, 701
coerceInt, 673
calledby coerceInt0, 674
calledby coerceInt1, 675
calledby coerceIntPermute, 685
calledby coerceIntSpecial, 692
calledby coerceIntX, 700
calledby coerceInt, 673
calledby retract2Specialization, 704
calledby retractUnderDomain, 709
calledby valueArgsEqual?, 696
calls coerceInt1, 673
calls coerceInt, 673
calls getMinimalVarMode, 673
calls objMode, 673
calls objVal, 673
calls unwrap, 673
defun, 673
coerceInt0, 674
calledby coerceInt0, 674
calledby coerceInteractive, 672
calls coerceInt0, 674
calls coerceInt, 674
calls conCoerceFrom, 674
calls intCodeGenCOERCCE, 674
calls isWrapped, 674
calls mkObj, 674
calls objMode, 674
calls objSetMode, 674
calls objVal, 674
calls unwrap, 674
uses $AnononyousFunction, 676
uses $Any, 676
uses $EmptyMode, 676
uses $Integer, 676
uses $NonNegativeInteger, 676
uses $OutputForm, 676
uses $PositiveInteger, 676
uses $OutputForm, 674
defun, 674
calledby coerceInt1, 675
calledby coerceInt, 673
calls NRTcompileEvalForm, 675
calls absolutelyCanCoerceByCheating, 675
calls asTupleAsList, 675
calls bottomUp, 675
calls coerceByFunction, 675
calls coerceInt1, 675
calls coerceInt2Union, 675
calls coerceIntAlgebraicConstant, 675
calls coerceIntFromUnion, 675
calls coerceIntTower, 675
calls coerceIntX, 675
calls coerceInt, 675
calls coerceRetract, 675
calls coerceSubDomain, 675
calls compareTypeLists, 675
calls deconstructT, 675
calls evalDomain, 675
calls getFunctionFromDomain, 675
calls getValue, 675
calls isEqualOrSubDomain, 675
calls isSubDomain, 675
calls mkAtreeNode, 675
calls mkAtree, 675
calls mkObjWrap, 675
calls mkObj, 675
calls nequal, 675
calls nreverse0, 676
calls objMode, 676
calls objVal, 676
calls selectLocalMms, 676
calls selectMms1, 676
calls transferPropsToNode, 676
calls unwrap, 676
uses $AnonymousFunction, 676
uses $Any, 676
uses $EmptyMode, 676
uses $Integer, 676
uses $NonNegativeInteger, 676
uses $OutputForm, 676
uses $PositiveInteger, 676
uses $QuotientField, 676
called by coerceIntTower, 682
calls coerceCommuteTest, 689
calls getl, 689
calls mkObjWrap, 689
calls objMode, 689
calls objValUnwrap, 689
calls strconc, 689
calls underDomainOf, 689
calls $coerceFailure, 689
catches, 689
defun, 689
correctInteractive, 672
called by coerceOrRetract, 703
called by coerceSpadArgs2E, 63
called by coerceSpadFunValue2E, 64
called by coerceTraceArgs2E, 62
called by coerceTraceFunValue2E, 64
called by interpret2, 280
calls clearDependentMaps, 672
calls coerectInt0, 672
calls mkObjWrap, 672
calls mkObj, 672
calls objMode, 672
calls objVal, 672
calls startTimingProcess, 672
calls stopTimingProcess, 672
calls throwKeyedMsg, 672
calls $EmptyMode, 672
calls $NoValueMode, 672
calls $OutputForm, 672
calls $compilingMap, 672
calls $insideCoerceInteractive, 672
calls $mapName, 672
defun, 672
correctIntFromUnion, 697
called by coerceInt1, 675
defun, 697
correctIntPermute, 685
called by coerceIntTower, 682
calls coercion, 685
calls computeTTTranspositions, 685
calls member, 685
calls objMode, 685
defun, 685
correctIntSpecial, 692
called by coerceIntTower, 682
INDEX

- calls coerceInt, 692
- calls objMode, 692
defun, 692

coerceIntTableOrFunction, 690
called by coerceIntTower, 682
calls assq, 690
calls coerceByFunction, 691
calls coerceByTable, 690
calls isLegitimateMode, 690
calls isValidType, 690
calls objMode, 690
calls objVal, 691
uses $CoerceTable, 691
defun, 690

call coerceIntTest, 684
called by coerceIntTower, 682
calls assq, 684
calls coerceConvertMmSelection, 684
uses $CoerceTable, 684
uses $useConvertForCoercions, 684
defun, 684

call coerceIntTower, 682
called by coerceInt1, 675
calls bubbleConstructor, 682
calls coerceIntByMap, 682
calls coerceIntCommute, 682
calls coerceIntPermute, 682
calls coerceIntSpecial, 682
calls coerceIntTableOrFunction, 682
calls coerceIntTest, 682
calls constructT, 682
calls deconstructT, 682
calls isValidType, 682
calls last, 682
calls replaceLast, 682
defun, 682

call coerceIntX, 700
called by coerceInt1, 675
calls coerceInt, 700
calls mkObjWrap, 700
calls underDomainOf, 700
calls unwrap, 700
defun, 700

call coerceOrRetract, 703
called by coerceOrThrowFailure, 695
called by evaluateType1, 1080
calls coerceInteractive, 703
calls retract, 703
defun, 703

call coerceOrThrowFailure, 695
called by coerceIntByMapInner, 694
calls coerceOrRetract, 695
calls coercionFailure, 695
calls mkObjWrap, 695
calls objValUnwrap, 695
defun, 695

call coerceRetract, 709
called by coerceInt1, 675
calls canFuncall?, 709
calls getI, 709
calls isEqualOrSubDomain, 709
calls mkObjWrap, 709
calls objMode, 709
calls objValUnwrap, 709
calls retractByFunction, 709
uses $Integer, 709
uses $OutputForm, 709
uses $SingleInteger, 709
uses $Symbol, 709
uses $coerceFailure, 709
catches, 709
defun, 709

call coerceSpadArgs2E, 63
called by coerceTraceArgs2E, 62
calls coerceInteractive, 63
calls exit, 63
calls mkObjWrap, 63
calls objValUnwrap, 63
calls seq, 63
uses $OutputForm, 63
uses $streamCount, 63
uses $tracedSpadModemap, 63
defun, 63

call coerceSpadFunValue2E, 64
called by coerceTraceFunValue2E, 64
calls coerceInteractive, 64
calls mkObjWrap, 64
calls objValUnwrap, 64
uses $OutputForm, 64
uses $streamCount, 64
uses $tracedSpadModemap, 64
defun, 64
coerceSubDomain,  700
    called by coerceInt1,  675
    called by coerceSubDomain,  701
    calls coerceImmediateSubDomain,  701
    calls coerceSubDomain,  701
    calls getdatabase,  700
    defun,  700
coerceTraceArgs2E,  61
    calls coerceInteractive,  62
    calls coerceSpadArgs2E,  62
    calls mkObjWrap,  62
    calls objValUnwrap,  62
    calls pname,  62
    calls spadsysnamep,  61
    uses $OutputForm,  62
    uses $mathTraceList,  62
    uses $tracedMapSignatures,  62
    defun,  61
coerceTraceFunValue2E,  64
    calls coerceInteractive,  64
    calls coerceSpadFunValue2E,  64
    calls lassoc,  64
    calls mkObjWrap,  64
    calls objValUnwrap,  64
    calls pname,  64
    calls spadsysnamep,  64
    uses $OutputForm,  64
    uses $mathTraceList,  64
    uses $tracedMapSignatures,  64
    defun,  61
collection,  611
    syntax,  611
commandAmbiguityError,  724
    called by commandErrorIfAmbiguous,  742
    called by commandErrorMessage,  721
    called by traceOptionError,  59
    called by userLevelErrorMessage,  722
    calls bright,  724
    calls sayKeyedMsg,  724
    calls sayMSG,  724
    calls terminateSystemCommand,  724
    defun,  724
commandError,  721
    called by commandErrorMessage,  721
    defun,  721
commandErrorIfAmbiguous,  742
    called by commandAmbiguityError,  742
    uses $oldline,  742
    uses line,  742
    defun,  742
commandErrorMessage,  721
    called by commandError,  721
    called by optionError,  721
    called by commandAmbiguityError,  721
    called by terminateSystemCommand,  722
    uses $oldline,  722
    uses line,  722
    defun,  721
commandsForUserLevel,  720
    called by synonymsForUserLevel,  1038
    called by systemCommand,  720
    called by unAbbreviateKeyword,  741
    called by whatCommands,  1093
    calls satisfiesUserLevel,  720
    defun,  720
commandUserLevelError,  722
    called by userLevelErrorMessage,  722
    defun,  722
compareposns,  592
    called by erMsgCompare,  592
    called by poCharPosn,  592
    called by poGlobalLinePosn,  592
    defun,  592
compareTypeLists,  700
    called by coerceInt1,  675
called by `setOutputAlgebra`, 967
called by `setOutputCharacters`, 971
called by `setOutputFormula`, 974
called by `setOutputHtml`, 980
called by `setOutputMathml`, 986
called by `setOutputOpenMath`, 990
called by `setOutputTex`, 1002
called by `showHistory`, 824
called by `spleI1`, 346
called by `summary`, 1035
called by `traceDomainConstructor`, 71
called by `traceReply`, 81
called by `undoCount`, 45
called by `untraceDomainConstructor`, 73
called by `writeInputLines`, 828
called by `xlSkip`, 314
calls `string-concatenate`, 1197
defun, 1197
`concatWithBlanks`, 1494
defun, 1494
`conCoerceFrom`
called by `coerceInt0`, 674
defun, 1446
`condErrorMsg`, 1446
defun, 1446
`conLowerCaseConTran`, 1494
called by `conLowerCaseConTran`, 1494
called by `conSpecialString?`, 1502
calls `hget`, 1197
calls `ifcar`, 1197
calls `ifcdr`, 1197
calls `isExposedConstructor`, 1197
calls `koPage`, 1503
calls `mkConform`, 1503
calls `ncParseFromString`, 1503
calls `opOf`, 1503
calls `Primitives`, 1503
defun, 1503
`conPage`, 1503
called by `constructorSearch`, 1501
called by `dbSelectCon`, 1552
called by `dbShowCons1`, 1547
calls `conPageFastPath`, 1503
calls `downcase`, 1503
calls `downlink`, 1503
calls `form2HtString`, 1503
calls `kPage`, 1503
calls `lassq`, 1503
calls `ySearch`, 1503
uses `$Primitives`, 1503
defun, 1503
`conPageChoose`, 1546
called by `dbShowCons1`, 1546
called by `getConstructorForm`, 1546
defun, 1546
`conPageConEntry`, 1504
called by `conPageFastPath`, 1504
calls `buildLlDbConEntry[9]`, 1504
uses `$conArgstrings`, 1504
defun, 1504
`conPageFastPath`, 1504
called by `conPageFastPath`, 1504
calls `charPosition`, 1504
uses `$conArgstrings`, 1504
uses `$conname`, 1504
uses `$doc`, 1504
uses `$exposed?`, 1504
uses `$kind`, 1504
defun, 1504
`conOpPage`, 1532
calls `conOpPage1`, 1532
calls `dbCompositeWithMap`, 1532
calls `dbExtractUnderlyingDomain`, 1532
calls `htProperty`, 1532
defun, 1532
`conOpPage1`, 1533
called by `conOpPage`, 1532
calls `capitalize`, 1533
calls `conPageFastPath`, 1533
calls `concat`, 1533
calls `dbSourceFile`, 1533
calls `dbSpecialOperations`, 1533
calls `dbXParts`, 1533
calls `htInitPage`, 1533
calls `htProperty`, 1533
calls `ifcar`, 1533
calls `ifcdr`, 1533
calls `isExposedConstructor`, 1533
calls `koPage`, 1533
calls `lassq`, 1533
calls `mkConform`, 1533
calls `ncParseFromString`, 1533
calls `opOf`, 1533
uses `$Primitives`, 1533
defun, 1533
`conPage`, 1503
called by `constructorSearch`, 1501
called by `dbSelectCon`, 1552
called by `dbShowCons1`, 1547
calls `conPageFastPath`, 1503
calls `downcase`, 1503
calls `downlink`, 1503
calls `form2HtString`, 1503
calls `kPage`, 1503
calls `lassq`, 1503
calls `ySearch`, 1503
uses `$Primitives`, 1503
defun, 1503
`conPageChoose`, 1546
called by `dbShowCons1`, 1546
called by `getConstructorForm`, 1546
defun, 1546
`conPageConEntry`, 1504
called by `conPageFastPath`, 1504
calls `buildLlDbConEntry[9]`, 1504
uses `$conArgstrings`, 1504
uses `$conArgstrings`, 1504
uses `$conname`, 1504
uses `$doc`, 1504
uses `$exposed?`, 1504
uses `$kind`, 1504
defun, 1504
`conPageFastPath`, 1504
called by `conPageFastPath`, 1504
calls `charPosition`, 1504
calls `conPageConEntry`, 1504
calls `dbRead`, 1504
calls `lassq`, 1504
INDEX

1667

calls length, 1504
uses $lowerCaseConTb, 1504
defun, 1504
consoleinput
   called by spad-syntax-error, 1115
conSpecialString?, 1502
called by conSpecialString?, 1502
called by constructorSearch, 1501
called by kArgumentCheck, 1529
calls conLowerCaseConTran, 1502
calls conSpecialString?, 1502
calls contained, 1502
calls dbString2Words, 1502
calls ifcar, 1502
calls kisValidType, 1502
calls member, 1502
calls ncParseFromString, 1502
calls strconc, 1502
calls ifcar, 1502
calls constructorSearchGrep
constructorSearchGrep
   called by constructorSearch, 1501
constructSubst, 664
called by hasAtt, 654
called by hasCaty, 650
called by spadTrace, 67
calls internl, 664
defun, 664
constructor
   called by coerceIntTower, 682
called by retractUnderDomain, 709
contains
   called by conSpecialString?, 1502
called by isPartialMode, 650
called by unifyStructVar, 659
containsVariables
   called by hasCate, 663
containsVars, 661
called by hasCateSpecial, 665
called by unifyStructVar, 659
calls containsVars1, 661
calls isPatternVar, 661
defun, 661
containsVars1, 661
called by containsVars1, 661
called by containsVars, 661
calls containsVars1, 661
calls isPatternVar, 661
defun, 661
copy
   called by makeInitialModemapFrame, 261
called by resetWorkspaceVariables, 894
called by undoSteps, 38
called by untrace, 59
copyright, 782, 787
calls concat, 787
calls getenviron, 787
calls obey, 787
defun, 787
manpage, 782
cot, 1221
defun, 1221
coth, 1223
defun, 1223
countCache, 912
called by setFunctionsCache, 911
calls identp, 912
calls insertAlist, 912
calls internal, 912
calls optionError, 912
calls qcar, 912
calls qcdr, 912
calls sayCacheCount, 912
calls sayKeyedMsg, 912
uses $cacheAlist, 912
uses $cacheCount, 913
uses $options, 912
defun, 912
createCurrentInterpreterFrame, 14
called by updateCurrentInterpreterFrame, 16
uses $HiFiAccess, 14
uses $HistListAct, 14
uses $HistListLen, 14
uses $HistList, 14
uses $HistRecord, 14
uses $IOindex, 14
uses $InteractiveFrame, 14
uses $InternalHistoryTable, 14
uses $InterpreterFrameName, 14
uses $localExposureData, 14
defun, 14
creditlist, 127
defvar, 127
credits, 788
used by credits, 788
uses credits, 788
defun, 788
manpage, 788
csc, 1222
defun, 1222
csch, 1223
defun, 1223
cSearch, 1499
calls constructorSearch, 1499
defun, 1499
curinstream, 245
used by ncIntLoop, 249
defvar, 245
curoutstream, 245
used by ncIntLoop, 249
defvar, 245
current-fragment
used by ioclear, 1118
current-line
used by ioclear, 1118
current-token
used by token-stack-show, 1117
currenttime
called by mkprompt, 267
DaaseName, 1169
called by interpopen, 1150
calls getenviron, 1169
uses $spadroot, 1169
defun, 1169
dalymode, 908
defvar, 908
dancols, 1226
defmacro, 1226
danrows, 1225
defmacro, 1225
daref2, 1225
defmacro, 1225
database, 1142
defstruct, 1142
dbAddChain, 1545
called by dbAddChain, 1545
called by dbSearchOrder, 1514
calls dbAddChainDomain, 1545
calls dbAddChain, 1545
defun, 1545
dbAddChainDomain, 1544
called by dbAddChain, 1545
calls dbInfovec, 1544
calls dbSubConform, 1544
calls devaluate, 1544
calls kFormatSlotDomain, 1544
uses $infovec, 1544
defun, 1544
dbAddDocTable, 1540
calledby dbDocTable, 1539
calls getConstructorForm, 1541
calls getdatabase, 1541
calls hget, 1541
calls lput, 1541
calls opOf, 1540
calls sublislis, 1541
uses $docTable, 1541
defun, 1540
dbCompositeWithMap, 1534
calledby conOpPage, 1532
calls dbExtractUnderlyingDomain, 1534
calls htpProperty, 1534
defun, 1534
dbConformGenUnder
calledby kPage, 1497
dbConsExposureMessage, 1549
calledby dbShowCons1, 1547
calls htpSay, 1549
uses $atLeastOneUnexposed, 1549
defun, 1549
dbConsHeading, 1553
calledby dbShowCons1, 1547
calls capitalize, 1553
calls form2HtString, 1553
calls htpProperty, 1553
calls length, 1553
calls member, 1553
calls unequal, 1553
calls pluralize, 1553
calls remdup, 1553
uses $exposedOnlyIfTrue, 1553
defun, 1553
dbConstructorDoc, 1539
calledby dbGetDocTable, 1543
calls dbConstructorDoc,fn, 1539
uses $op, 1539
uses $sig, 1539
defun, 1539
dbConstructorDoc, gn, 1539
calledby dbConstructorDoc,ln, 1539
calls length, 1538
calls sublislis, 1538
uses $FormalMapVariableList, 1538
uses $op, 1539
uses $sig, 1538
defun, 1538

dbConstructorDoc, ln, 1538
calledby dbConstructorDoc,gn, 1539
calls length, 1538
calls sublislis, 1538
uses $FormalMapVariableList, 1538
uses $op, 1539
uses $sig, 1538
defun, 1538
dbConstructorKind
calledby dbShowCons1, 1547
dbDelimiters, 1495
devar, 1495
dbDocTable, 1539
calls dbAddDocTable, 1539
calls hget, 1539
calls make-hashtable, 1539
calls originsInOrder, 1539
uses $docTableHash, 1539
uses $docTable, 1539
defun, 1539
dbExtractUnderlyingDomain, 1535
calledby conOpPage, 1532
calledby dbCompositeWithMap, 1534
calls isValidType, 1535
defun, 1535
dbGatherData
calledby kePageDisplay, 1512
dbGetDocTable, 1542
calls dbConstructorDoc, 1543
calls dbGetDocTable,gn, 1543
calls hget, 1543
calls qcdr, 1543
calls string2Integer, 1542
uses $conform, 1543
uses $op, 1543
uses $sig, 1543
uses $which, 1543
defun, 1542
dbGetDocTable,gn, 1542
calledby dbGetDocTable, 1543

calls dbGetDocTable,hn, 1542
calls lastatom, 1542
uses $conform, 1542
defun, 1542
dbGetDocTable,hn, 1541
calledby dbGetDocTable,gn, 1542
calls qcar, 1541
calls qcdr, 1541
calls sublis, 1541
uses $FormalMapVariableList, 1541
uses $conform, 1541
uses $sig, 1541
uses $which, 1541
defun, 1541
dbGetInputString
calledby dbShowCons, 1545
calledby konPageFilterByName, 1538
dbInfovec
calledby dbAddChainDomain, 1544
calledby dbSearchOrder, 1514
dbKind
calledby constructorSearch, 1501
dbMkEvalable, 1529
calledby kDomainName, 1528
calls getdatabase, 1529
calls mkEvalable, 1529
defun, 1529
dbMkForm, 1562
defun, 1562
dbName
calledby constructorSearch, 1501
calledby dbShowConstructorLines, 1554
dbNonEmptyPattern, 1476
defun, 1476
dbPart
calledby alqlGetKindString, 1236
calledby alqlGetOrigin, 1235
calledby alqlGetParams, 1236
dbpHasDefaultCategory?
calledby kcPage, 1515
dbPresentCons
calledby dbShowCons1, 1547
dbRead, 1500
calledby conPageFastPath, 1504
defun, 1500
dbSayItems
calledby dbShowConditions, 1552
dbSearchOrder, 1514
calledby ksPage, 1513
calls dbAddPage, 1514
calls dbInfovec, 1514
calls dbSubConform, 1514
calls devaluate, 1514
calls getdatabase, 1514
calls kFormatSlotDomain, 1514
calls kTestPred, 1514
calls opOf, 1514
calls simpCatPredicate, 1514
calls sublis, 1514
uses $domain, 1514
uses $infovec, 1514
uses $predvec, 1514
defun, 1514
dbSelectCon, 1552
calledby conPage, 1552
calls htpProperty, 1552
calls opOf, 1552
defun, 1552
dbShowConditions, 1552
calledby dbShowCons1, 1547
calls bcConPredTable, 1552
calls dbSayItems, 1552
calls htpTable, 1552
calls length, 1552
calls opOf, 1552
calls pluralize, 1552
calls splitConTable, 1552
defun, 1552
dbShowCons, 1545
calledby dbShowConsKindsFilter, 1549
calledby dbShowCons, 1545
calledby kArgPage, 1506
calledby kcaPage1, 1521
calledby kccPage, 1522
calledby kcdePage, 1524
calledby kcnuPage, 1526
calledby kepPage, 1519
calledby kcuPage, 1525
calledby ksPage, 1513
calls bcErrorPage, 1545
calls constructor?, 1545
calls dbGetInputString, 1545
calls dbShowCons1, 1545
calls dbShowCons, 1545
calls downcase, 1545
calls emptySearchPage, 1545
calls htCopyPropList, 1545
calls htInitPageNoScroll, 1545
calls htpProperty, 1545
calls htpSetProperty, 1545
calls ifcar, 1545
calls member, 1545
calls pmTransFilter, 1545
calls superMatch?, 1545
uses $exposedOnlyIfTrue, 1545
defun, 1545
dbShowCons1, 1547
calledby conPageChoose, 1546
calledby dbShowConsLine, 1546
calledby dbShowCons, 1545
calledby assocleft, 1547
calls bcAbbTable, 1547
calls bcConTable, 1547
calls bcNameConTable, 1547
calls bcUnixTable, 1547
calls conPage, 1547
calls dbConsExposureMessage, 1547
calls dbConsHeading, 1547
calls dbConstructorKind, 1547
calls dbPresentCons, 1547
calls dbShowConditions, 1547
calls dbShowConsDoc, 1547
calls dbShowConsKinds, 1547
calls function, 1547
calls getCDTEntry, 1547
calls getTable, 1547
calls htCopyPropList, 1547
calls htInitPageNoScroll, 1547
calls htpSetProperty, 1547
calls displayDomainOp, 1550
calls getConstructorForm, 1550
calls getConstructorSignature, 1550
calls getdatabase, 1550
calls getl, 1550
calls intern, 1550
calls isExposedConstructor, 1550
calls member, 1550
calls sublisFormal, 1550
calls sublis, 1550
calls systemError, 1549
defun, 1549
dbShowConsDoc, 1549
calledby dbShowCons1, 1547
calledby dbShowConsDoc, 1549
calledby dbShowConsDoc1, 1549
calledby kPage, 1497
calledby kiPage, 1509
calls displayDomainOp, 1550
calls getConstructorForm, 1549
calls getConstructorSignature, 1550
calls getdatabase, 1550
calls getl, 1550
calls intern, 1550
calls isExposedConstructor, 1550
calls member, 1550
calls sublisFormal, 1550
calls sublis, 1550
calls systemError, 1549
defun, 1549
dbShowConsDoc1, 1550
calledby dbShowConsDoc, 1549
calledby dbSpecialDescription, 1555
calledby kPage, 1497
calledby kiPage, 1509
calledby kPage, 1497
calledby klPage, 1509
calls displayDomainOp, 1550
calls getConstructorForm, 1549
calls getConstructorSignature, 1550
calls getdatabase, 1550
calls getl, 1550
calls intern, 1550
calls systemError, 1549
defun, 1550
dbShowConsKinds, 1547
calledby dbShowCons1, 1547
dbShowConsKindsFilter, 1549
calledby dbShowCons, 1549
calledby dbShowCons1, 1554
calledby function, 1554
calls getConstructorForm, 1554
calls getl, 1554
calls intern, 1554
calls listSort, 1554
defun, 1554
dbShowOp1
called by dbSpecialOperations, 1556
dbShowOperationsFromConform
called by koPageAux1, 1537
called by koPageAux, 1537
dbSourceFile
called by conOpPage1, 1533
called by kPage, 1497
dbSowOpItems
called by kePageDisplay, 1512
dbSpecialDescription, 1555
calls dbShowConsDoc1, 1555
calls form2HtString, 1555
calls getConstructorForm, 1555
calls htInitPage, 1555
calls htShowPage, 1555
calls htpSetProperty, 1555
uses $conformsAreDomains, 1555
defun, 1555
dbSpecialExpandIfNecessary, 1557
called by dbSpecialExports, 1556
called by dbSpecialOperations, 1556
calls qcadar, 1557
calls qcar, 1557
calls qcdar, 1557
calls qcdr, 1557
defun, 1557
dbSpecialExports, 1556
calls dbSpecialExpandIfNecessary, 1556
calls form2HtString, 1556
calls getConstructorForm, 1556
calls getl, 1556
calls htInitPage, 1556
calls htShowPage, 1556
calls kePageDisplay, 1556
defun, 1556
dbSpecialOperations, 1556
called by conOpPage1, 1533
called by dbShowOp1, 1556
calls dbSpecialExpandIfNecessary, 1556
calls form2HtString, 1556
calls getConstructorForm, 1556
calls getl, 1556
calls htInitPage, 1556
calls htpSetProperty, 1556
defun, 1556
dbSplitLibdb
called by make-databases, 1165
dbString2Words, 1495
called by conSpecialString?, 1502
calls dbWordFrom, 1495
defun, 1495
dbSubConform, 1544
called by dbAddChainDomain, 1544
called by dbSearchOrder, 1514
called by dbSubConform, 1544
calls dbSubConform, 1544
calls position, 1544
uses $FormalMapVariableList, 1544
defun, 1544
dbWordFrom, 1496
called by dbString2Words, 1495
calls maxindex, 1496
calls member, 1496
calls strconc, 1496
uses $dbDelimiters, 1496
defun, 1496
dbXParts
called by conOpPage1, 1533
called by kPage, 1497
dc1
called by reportOpsFromLisplib, 1025
debugmode
used by spad-syntax-error, 1115
decideHowMuch, 581
called by getPosStL, 578
calls poFileName, 581
calls poLinePosn, 582
calls poNopos?, 581
calls poPosImmediate?, 581
defun, 581
decomposeTypeIntoTower, 688
called by computeTTTranspositions, 686
called by decomposeTypeIntoTower, 688
calls decomposeTypeIntoTower, 688
calls deconstructT, 688
defun, 688
decomposeT

called by absolutelyCanCoerceByCheating, 702
called by coerceCommuteTest, 690
called by coerceInt1, 675
called by coerceIntByMap, 693
called by coerceIntTower, 682
called by decomposeTypeIntoTower, 688
called by retractUnderDomain, 709
defaultTargetFE, 667
called by defaultTargetFE, 668
called by hasCateSpecialNew, 666
calls defaultTargetFE, 668
calls ifcar, 668
calls isEqualOrSubDomain, 667
local ref $FunctionalExpression, 668
local ref $Integer, 668
local ref $RationalNumber, 668
local ref $Symbol, 668
defun, 667
defstream, 1128
called by reportOpsFromLisplib1, 1024
called by reportOpsFromUnitDirectly1, 1032
called by sayMSG2File, 28
called by setOutputAlgebra, 967
called by setOutputFormula, 995
called by setOutputFortran, 974
called by setOutputHtml, 980
called by setOutputMathml, 986
called by setOutputOpenMath, 990
called by setOutputTex, 1001
called by writeInputLines, 828
defun, 1128
defmacro
ancolsU16, 1252
ancolsU32, 1254
ancolsU8, 1251
anrowsU16, 1252
anrowsU32, 1254
anrowsU8, 1250
aref2U16, 1251
aref2U32, 1253
aref2U8, 1250
assq, 1200
bit-to-truth, 1232
bvec-elt, 1232
bvec-setelt, 1232
bvec-size, 1233
cdancols, 1208
cdanrows, 1208
cdaref2, 1207
cdelt, 1209
cdlen, 1210
cdsetaref2, 1207
cdsetelt, 1209
dancols, 1226
danrows, 1225
daref2, 1225
delt, 1227
DFAcos, 1216
DFAcosh, 1219
DFAdd, 1212
DFAsin, 1216
DFAsinh, 1218
DFAatan, 1217
DFAatan2, 1217
DFAatanh, 1219
DFCos, 1216
DFCosh, 1218
DFDivide, 1213
DFEql, 1213
DFExp, 1215
DFExpt, 1215
DFIntegerDivide, 1214
DFIntegerExpt, 1215
DFIntegerMultiply, 1212
DFLessThan, 1211
DFLog, 1214
DFLogE, 1214
DFMax, 1213
DFMin, 1213
DFMinusp, 1211
DFMultiply, 1212
DFSin, 1215
DFSinh, 1217
DFSqrt, 1214
DFSubtract, 1212
DFTan, 1216
DFTanh, 1218
DFUnaryMinus, 1211
DFZerop, 1211
dlen, 1226
dsetaref2, 1225
dsetelt, 1227
eU16, 1248
eU32, 1249
INDEX

/tracereply, 77
?t, 84
abbrQuery, 797
abbreviations, 754
abbreviationsSpad2Cmd, 754
absolutelyCanCoerceByCheating, 702
acot, 1221
acoth, 1224
asec, 1222
asech, 1223
addBinding, 1109
addBindingInteractive, 1113
addInputLibrary, 907
addNewInterpreterFrame, 18
addoperations, 1153
addTraceItem, 83
algCoerceInteractive, 1241
algEqual, 696
allConstructors, 1178
allOperations, 1179
alqlGetKindString, 1236
alqlGetOrigin, 1235
alqlGetParams, 1236
alreadyOpened?, 585
apropos, 1097
asec, 1222
asech, 1224
assertCond, 321
augmentHasArgs, 1523
augmentTraceNames, 65
axiomVersion, 748
basicLookup, 1185
basicLookupCheckDefaults, 1188
basicStringize, 1415
bcComplexLimit, 1377
bcComplexLimitGen, 1379
bcCreateVariableString, 1387
bcDefiniteIntegrate, 1344
bcDefiniteIntegrateGen, 1345
bcDifferentiate, 1349
bcDifferentiateGen, 1350
bcDraw, 1351
bcDraw2Dfun, 1352
bcDraw2DfunGen, 1353
bcDraw2Dpar, 1354
bcDraw2DparGen, 1355
bcDraw2DSolve, 1355
bcDraw2DSolveGen, 1357
bcDraw3Dfun, 1357
bcDraw3DfunGen, 1359
bcDraw3Dpar, 1360
bcDraw3Dpar1, 1362
bcDraw3Dpar1Gen, 1363
bcDraw3DparGen, 1361
bcDrawIt, 1401
bcDrawIt2, 1342
bcError, 1401
bcFindString, 1398
bcFinish, 1397
bcGen, 1398, 1399
bcGenEquations, 1396
bcGenExplicitMatrix, 1341
bcHt, 1414
bchtMakeButton, 1440
bcIndefiniteIntegrate, 1343
bcIndefiniteIntegrateGen, 1344
bcInputEquations, 1385
bcInputEquationsEnd, 1389
bcInputExplicitMatrix, 1339
bcInputMatrixByFormula, 1337
bcInputMatrixByFormulaGen, 1338
bcInputSolveInfo, 1384
bcIssueHt, 1414
bcLaurentSeries, 1369
bcLaurentSeriesGen, 1373
bcLimit, 1374
bcLinearExtractMatrix, 1392
bcLinearMatrixGen, 1394
bcLinearSolve, 1381
bcLinearSolveEqns, 1382
bcLinearSolveEqns1, 1384
bcLinearSolveEqnsGen, 1396
bcLinearSolveMatrix, 1391
bcLinearSolveMatrix1, 1391
bcLinearSolveMatrixHomo, 1394
bcLinearSolveMatrixInhomo, 1392
bcLinearSolveMatrixInhomoGen, 1393
bcMakeEquations, 1388
bcMakeLinearEquations, 1388
bcMakeUnknowns, 1387
bcMatrix, 1336
bcMatrixGen, 1341
bcMkFunction, 1397
bcNotReady, 1401
bcOptional, 1399
bcProduct, 1348
bcProductGen, 1348
bcPuiseuxSeries, 1371
bcPuiseuxSeriesGen, 1373
bcReadMatrix, 1336
bcRealLimit, 1375
bcRealLimitGen, 1376
bcRealLimitGen1, 1377
bcSadFaces, 1422
bcSeries, 1364
bcSeriesByFormula, 1366
bcSeriesByFormulaGen, 1369
bcSeriesExpansion, 1365
bcSeriesExpansionGen, 1366
bcSeriesGen, 1373
bcSolve, 1380
bcSolveEquations, 1390
bcSolveEquationsNumerically, 1389
bcSolveNumerically1, 1390
bcSolveSingle, 1383
bcString2HyString, 1398
bcString2HyString2, 1398
bcString2WordList, 1400
bcSum, 1346
bcSumGen, 1347
bcSystemSolve, 1382
bcSystemSolveEqns1, 1384
bcTaylorSeries, 1367
bcTaylorSeriesGen, 1372
bcUnixTable, 1554
bcVectorGen, 1400
bcvspace, 1400
bcwords2liststring, 1400
beforeAfter, 1425
BooleanEquality, 1206
bracketString, 1449
break, 87
breaklet, 86
brightprint, 1198
brightprint-0, 1198
browseopen, 1151
buildHtMacroTable, 1333
buttonNames, 1429
bvec-and, 1234
bvec-concat, 1233
bvec-copy, 1233
bvec-equal, 1233
bvec-greater, 1233
bvec-make-full, 1232
bvec-nand, 1234
bvec-nor, 1235
bvec-not, 1235
bvec-or, 1234
bvec-xor, 1234
canFuncall?, 1197
catchCoerceFailure, 692
categoryopen, 1152
changeHistListLen, 830
changeToNamedInterpreterFrame, 17
charDigitVal, 859
checkCondition, 1445
checkFilter, 1493
chkAllNonNegativeInteger, 1467
chkDirectory, 1466
chkNameList, 1465
chkNonNegativeInteger, 1467
chkOutputFileName, 1466
chkPosInteger, 1466
chkRange, 1467
cleanline, 791
cleanupLine, 801
clear, 767
clearCmdAll, 771
clearCmdCompletely, 770
clearCmdExcept, 772
clearCmdParts, 773
clearCmdSortedCaches, 769
clearFrame, 1088
clearMacroTable, 772
clearParserMacro, 725
clearSpad2Cmd, 767
close, 777
closeInterpreterFrame, 22
cmpnote, 251
coerceBranch2Union, 698
coerceByFunction, 680
coerceByTable, 691
coerceCommuteTest, 690
coerceConvertMmSelection, 684
INDEX

coerceImmediateSubDomain, 701
c coerceInt, 673
c coerceInt0, 674
c coerceInt1, 675
c coerceInt2Union, 697
c coerceIntAlgebraicConstant, 698
c coerceIntByMap, 693
c coerceIntByMapInner, 694
c coerceIntCommute, 689
c coerceInteractive, 672
c coerceIntFromUnion, 697
c coerceIntPermute, 685
c coerceIntSpecial, 692
c coerceIntTableOrFunction, 690
c coerceIntTest, 684
c coerceIntTower, 682
c coerceIntX, 700
c coerceOrRetract, 703
c coerceOrThrowFailure, 695
c coerceRetract, 709
c coerceSpadArgs2E, 63
c coerceSpadFunValue2E, 64
c coerceSubDomain, 700
c coerceTraceArgs2E, 61
c coerceTraceFunValue2E, 64
c coerceUnion2Branch, 707
c coercionFailure, 695
c coerceSpecialString?, 1502
c constoken, 337
c constructorSearch, 1501
c containsVars, 661
c containsVars1, 661
c copyright, 787
c cot, 1221
c coth, 1223
c countCache, 912
c createCurrentInterpreterFrame, 14
c credits, 788
c csc, 1222
c csch, 1223
c Search, 1499
DaaseName, 1169
dbAddChain, 1545
dbAddChainDomain, 1544
dbAddDocTable, 1540
dbCompositeWithMap, 1534
dbConsExposureMessage, 1549
dbConsHeading, 1553
dbConstructorDoc, 1539
dbConstructorDoc.gn, 1539
dbConstructorDoc.hn, 1538
dbDocTable, 1539
dbExtractUnderlyingDomain, 1535
dbGetDocTable, 1542
dbGetDocTable.gn, 1542
dbGetDocTable.hn, 1541
dbMkEvalable, 1529
dbMkForm, 1562
dbNonEmptyPattern, 1476
dbRead, 1500
dbSearchOrder, 1514
dbSelectCon, 1552
dbShowConditions, 1555
dbShowCons, 1545
dbShowCons1, 1547
dbShowConsDoc, 1549
dbShowConsDoc1, 1550
dbShowConsKindsFilter, 1549
hasCate, 663
hasCate1, 657
hasCateSpecial, 665
hasCateSpecialNew, 666
hasCatExpression, 657
hasCaty, 650
hasCaty1, 662
hasCorrectTarget, 685
hashable, 1246
hasOptArgs?, 539
hasOption, 723
hasPair, 1067
hasSig, 653
hasSigAnd, 655
hasSigOr, 656
help, 812
helpSpad2Cmd, 813
histFileErase, 860
histFileName, 820
histInputFileName, 820
history, 821
historySpad2Cmd, 822
hkeys, 1195
hput, 1194
htAddHeading, 1415
htAllOrNum, 1474
htBcLinks, 1423
htBcLispLinks, 1424
htBcRadioButtons, 1427
htCacheAddChoice, 1471
htCacheOne, 1474
htCacheSet, 1473
htCheck, 1463
htCheckList, 1464
htDoneButton, 1442
htDoNothing, 1463
htEscapeString, 1450
htFunctionSetLiteral, 1460
htGloss, 1482
htGlossPage, 1482
htGlossSearch, 1486
htGreekSearch, 1487
htInitPage, 1415
htInputStrings, 1429
htKill, 1461
htLispLinks, 1422
htLispMemoLinks, 1423
htMakeButton, 1438
htMakeDoitButton, 1441
htMakeDoneButton, 1437
htMakeErrorPage, 1418
htMakeInputList, 1448
htMakeLabel, 1472
htMakePage, 1416
htMakePage1, 1417
htMakePathKey, 1468
htMakePathKey.fn, 1468
htMakeTemplates, 1436
htMakeTemplates, substLabel, 1435
htMarkTree, 1468
ht MkName, 1402
htpAddInputAreaProp, 1408
htpAddToPageDescription, 1413
htpButtonValue, 1406
htpDestroyPage, 1403
htpDomainConditions, 1405
htpDomainPvarSubstList, 1406
htpDomainVariableAlist, 1405
htpInputAreaAlist, 1407
htpLabelDefault, 1411
htpLabelErrorMsg, 1411
htpLabelFilter, 1412
htpLabelFilteredInputString, 1409
htpLabelInputString, 1409
htpLabelSpadType, 1412
htpLabelSpadValue, 1410
htpLabelType, 1411
htpName, 1404
htpPageDescription, 1412
htpProperty, 1408
htpPropertyList, 1408
htpRadioButtonAlist, 1406
htProcessBcButtons, 1420
htProcessBcStrings, 1421
htProcessDoitButton, 1441
htProcessDomainConditions, 1431
htProcessDoneButton, 1438
htProcessToggleButtons, 1419
htSetDomainConditions, 1405
htSetDomainPvarSubstList, 1406
htSetDomainVariableAlist, 1405
htSetInputAreaAlist, 1407
htpSetLabelErrorMsg, 1411
htpSetLabelInputString, 1410
htpSetLabelSpadValue, 1410
htpSetName, 1404
htpSetPageDescription, 1413
htpSetProperty, 1404
htpSetRadioButtonAlist, 1407
htQuote, 1418
htRadioButtons, 1426
htSetCache, 1470
htSetExpose, 1470
htSetFunCommand, 1461
htSetFunCommandContinue, 1461
htSetHistory, 1469
htSetInputLibrary, 1469
htSetInteger, 1458
htSetLinkerArgs, 1470
htSetLiteral, 1456
htSetLiterals, 1455
htSetNotAvailable, 1462
htSetOutputCharacters, 1470
htSetOutputLibrary, 1469
htSetSystemVariable, 1482
htSetSystemVariableKind, 1481
htSetvarDoneButton, 1460
htSetVars, 1450
htShowCount, 1453
htShowFunctionPage, 1458
htShowFunctionPageContinued, 1459
htShowIntegerPage, 1457
htShowLiteralsPage, 1455
htShowPage, 1416
htShowPageNoScroll, 1416
htShowSetPage, 1454
htShowSetTree, 1451
htShowSetTreeValue, 1453
htStringPad, 1402
htsv, 1450
htSystemVariables, 1479
htSystemVariables,displayOptions, 1477
htSystemVariables,fn, 1477
htSystemVariables,functionTail, 1478
htSystemVariables,gn, 1476
htTextSearch, 1489
htTutorialSearch, 1491
 If?, 304
ifCond, 314
ignorep, 888
iht, 1413
importFromFrame, 19
incActive?, 328
incAppend, 312
incAppend1, 312
incBiteOff, 862
incClassify, 324
incCommand?, 326
incCommandTail, 327
incConsoleInput, 328
incDrop, 327
incFileInput, 327
incFileName, 862
incHandleMessage, 301
incIgen, 299
incIgen1, 300
incIframe, 327
incLine, 312
incLine1, 313
inclmsgCannotRead, 317
inclmsgCmdBug, 323
inclmsgConActive, 318
inclmsgConsole, 320
inclmsgConStill, 319
inclmsgFileCycle, 317
inclmsgFinSkipped, 320
inclmsgIfBug, 323
inclmsgIfSyntax, 322
inclmsgNoSuchFile, 316
inclmsgPrematureEOF, 313
inclmsgPrematureFin, 321
inclmsgSay, 315
incLude, 301
incLude1, 306
incNConsoles, 328
incPrefix?, 326
incRenumber, 298
incRenumberItem, 300
incRenumberLine, 300
incRgen, 329
incRgen1, 330
incStream, 298
incString, 263
incZip, 298
incZip1, 299
init-boot/spad-reader, 1114
init-memory-config, 258
initHist, 820
initHistList, 821
initial-getdatabase, 1147
initializeInterpreterFrameRing, 12
initializeSetVariables, 894
initImPr, 589
initroot, 259
initToWhere, 590
insertAlist, 913
insertpile, 557
insertPos, 601
integer-decode-float-denominator, 1220
integer-decode-float-exponent, 1220
integer-decode-float-numerator, 1219
integer-decode-float-sign, 1220
InterpExecuteSpadSystemCommand, 256
interpFunctionDepAlists, 737
interpopen, 1149
interpret, 278
interpret1, 279
interpret2, 280
interpretTopLevel, 278
intInterpretPform, 292
intloop, 250
intloopEchoParse, 294
intloopInclude, 287
intloopInclude0, 288
intloopPrefix?, 259
intloopProcess, 288
intloopProcessString, 261
intloopReadConsole, 253
intloopSpadProcess, 289
intloopSpadProcess,interp, 290
intunlisp, 260
intProcessSynonyms, 257
ioclear, 1118
iostat, 1116
isDomainOrPackage, 1061
isDomainValuedVariable, 1105
isEqualOrSubDomain, 668
isExposedConstructor, 1026
isgenvar, 1064
isIntegerString, 740
isInterpOnlyMap, 1058
isListOfIdentifiers, 1055
isListOfIdentifiersOrStrings, 1056
isPartialMode, 650
isPatternVar, 661
isSharpVar, 1064
isSharpVarWithNum, 1063
isSubForRedundantMapName, 1059
isSystemDirectory, 1183
isTaggedUnion, 1077
isTraceGensym, 1061
isUncompiledMap, 1058
justifyMyType, 286
kArgPage, 1506
kArgumentCheck, 1529
kcaPage, 1520
kcaPage1, 1521
kccPage, 1522
kcedePage, 1524
kdecodePage, 1521
kdecodePage, 1520
kCheckArgumentNumbers, 1531
kcnPage, 1526
kcPage, 1515
kcpPage, 1518
kcuPage, 1525
kDomainName, 1528
kdPageInfo, 1505
KeepPart?, 305
kePage, 1509
kePageDisplay, 1512
kePageOpAlist, 1511
keyword, 344
keyword?, 344
kiPage, 1508
kisValidType, 1530
kooPageFilterByName, 1537
koPage, 1535
koPageAux, 1536
koPageAux1, 1537
koPageFromKkPage, 1536
koPageInputAreaUnchanged?, 1527
kSearch, 1500
ksPage, 1513
kTestPred, 1543
lassocSub, 1057
INDEX

lastcount, 888
lastTokPosn, 563
leaveScratchpad, 873
lefts, 1562
letPrint, 1062
letPrint2, 1064
letPrint3, 1066
lfcomment, 340
lferror, 359
lffloat, 351
lfid, 338
lfinteger, 357
lfkey, 345
lfnegcomment, 341
lfrinteger, 357
lfspaces, 353
lfstring, 353
libConstructorSig, 1563
libdbTrim, 1500
library, 1159
license, 866
linearFinalRequest, 1395
lineoftoks, 334
listConstructorAbbreviations, 756
listDecideHowMuch, 583
listOfStrings2String, 1454
listOutputter, 576
lnCreate, 567
lnExtraBlanks, 567
lnFileName, 569
lnFileName?, 569
lnGlobalNum, 568
lnImmediate?, 569
lnLocalNum, 568
lnPlaceOfOrigin, 568
lnSetGlobalNum, 568
lnString, 567
loadFunctor, 1184
loadLib, 1181
loadLibNoUpdate, 1183
localdatabase, 1160
localnrlib, 1161
lookupInDomainVector, 1187
loopIters2Sex, 541
lotsof, 1194
ltrace, 868
mac0Define, 458
mac0ExpandBody, 453
mac0Get, 456
mac0GetName, 454
mac0InfiniteExpansion, 453
mac0InfiniteExpansion,name, 454
mac0MLambdaApply, 452
mac0SubstituteOuter, 459
macApplication, 451
macExpand, 450
macId, 455
macLambda, 457
macLambda,mac, 457
macLambdaParameterHandling, 459
macMacro, 457
macroExpanded, 450
macSubstitutedId, 460
macSubstituteOuter, 459
macWhere, 456
macWhere,mac, 456
make-absolute-filename, 260
make-appendstream, 1128
make-databases, 1164
make-istream, 1127
make-outstream, 1127
makeByteWordVec2, 1305
makeFullNamestring, 1131
makeHistFileName, 819
makeInitialModemapFrame, 261
makeInputFilename, 1129
makeLeaderMsg, 599
makeMsgFromFile, 593
makeOrdinal, 1083
makePathname, 1193
makeSpadCommand, 1448
makeStream, 1129
manexp, 1220
mapLetPrint, 74
mapStringize, 1414
member, 1198
mergePathnames, 1191
messageprint, 1199
messageprint-1, 1199
messageprint-2, 1199
mkConform, 1532
mkCurryFun, 1425
mkDomPvar, 663
mkDomTypeForm, 1507
mkEvalable, 1075
mkEvalableMapping, 1078
mkEvalableRecord, 1078
mkEvalableUnion, 1077
mkLineList, 295
mkobjFn, 1237
mkprompt, 266
mkSetTitle, 1454
mkUnixPattern, 1492
monitor-add, 1316
monitor-apropos, 1329
monitor-autoload, 1324
monitor-checkpoint, 1321
monitor-decr, 1318
monitor-delete, 1316
monitor-dirmame, 1324
monitor-disable, 1317
monitor-enable, 1316
monitor-end, 1315
monitor-exposedp, 1326
monitor-file, 1319
monitor-help, 1322
monitor-increment, 1318
monitor-info, 1319
monitor-inittable, 1314
monitor-libname, 1325
monitor-nrlib, 1325
monitor-parse, 1328
monitor-percent, 1329
monitor-readinterp, 1326
monitor-report, 1327
monitor-reset, 1317
monitor-restore, 1321
monitor-results, 1315
monitor-spaddir, 1328
monitor-tested, 1320
monitor-untested, 1319
monitor-write, 1320
msgCreate, 569
msgImPr?, 580
msgNoRep?, 597
msgOutputter, 575
msgText, 286
myWritable?, 1231
namestring, 1190
ncAlist, 637
ncBug, 590
ncConversationPhase, 292
ncConversationPhase, wrapup, 293
ncElitQ, 638
ncError, 293
ncHardError, 574
ncIntLoop, 249
ncloopCommand, 750
ncloopDQlines, 296
ncloopEscaped, 261
ncloopIncFileName, 861
ncloopInclude, 862
ncloopInclude0, 297
ncloopInclude1, 861
ncloopParse, 262
ncloopPrefix?, 750
ncloopPrintLines, 294
ncParseFromstring, 1240
ncPutQ, 638
ncSoftError, 574
ncTag, 637
ncTopLevel, 248
newHelpSpad2Cmd, 813
next, 262
next-lines-clear, 1118
next-lines-show, 1117
next1, 263
nextInterpreterFrame, 17
nextline, 336
nonBlank, 295
npADD, 382
npAdd, 383
npAmpersand, 428
npAmpersandFrom, 427
npAndOr, 406
npAngleBared, 411
npAnyNo, 387
npApplication, 386
npApplication2, 387
npArith, 425
npAssign, 441
npAssignment, 441
npAssignVariable, 442
npAtom1, 407
| npAtom2    | 384 |
| npBacksetElse | 421 |
| npBackTrack  | 372 |
| npBDefinition | 409 |
| npboot      | 744 |
| npBPileDefinition | 412 |
| npBraced    | 411 |
| npBracked   | 410 |
| npBracketed | 410 |
| npBreak     | 399 |
| npBy        | 424 |
| npCategory  | 377 |
| npCategoryL | 377 |
| npCoerceTo  | 444 |
| npColon     | 442 |
| npColonQuery | 444 |
| npComma     | 370 |
| npCommaBackSet | 370 |
| npCompMissing | 375 |
| npConditional | 420 |
| npConditionalStatement | 404 |
| npConstTok  | 408 |
| npDDInfKey  | 432 |
| npDecl      | 439 |
| npDef       | 412 |
| npDefaultDecl | 394 |
| npDefaultItem | 394 |
| npDefaultItemlist | 393 |
| npDefaultValue | 419 |
| npDefinition | 391 |
| npDefinitionItem | 392 |
| npDefinitionlist | 417 |
| npDefinitionOrStatement | 372 |
| npDefn      | 411 |
| npDefTail   | 419 |
| npDiscrim   | 422 |
| npDisjand   | 422 |
| npDollar    | 408 |
| npDotted    | 386 |
| npElse      | 420 |
| npEncAp     | 406 |
| npEncl      | 407 |
| npEnclosed  | 436 |
| npEqKey     | 369 |
| npEq Peek   | 376 |
| npExit      | 440 |
| npExport    | 395 |
| npExpress   | 403 |
| npExpress1  | 404 |
| npFirstTok  | 367 |
| npFix       | 390 |
| npForIn     | 402 |
| npFree      | 398 |
| npFromdom   | 427 |
| npFromdom1  | 428 |
| npGives     | 372 |
| npId        | 429 |
| npImport    | 405 |
| npInGeneric | 432 |
| npInfixOp   | 385 |
| npInfixOperator | 384 |
| npInfKey    | 433 |
| npInline    | 398 |
| npInterval  | 424 |
| npItem      | 366 |
| npItem1     | 366 |
| npIterate   | 399 |
| npIterator  | 400 |
| npIterators | 400 |
| npLambda    | 373 |
| npLeftAssoc | 431 |
| npLet       | 390 |
| npLetQualified | 391 |
| npLisp      | 744 |
| npList      | 379 |
| npListAndRecover | 414 |
| npListing   | 379 |
| npListOfFun | 445 |
| npLocal     | 397 |
| npLocalDecl | 397 |
| npLocalItem | 397 |
| npLocalItemlist | 396 |
| npLogical   | 421 |
| npLoop      | 399 |
| npMacro     | 388 |
| npMatch     | 374 |
| npMDEF      | 389 |
| npMdef      | 389 |
| npMDEFinition | 390 |
| npMissing   | 375 |
| npMissingMate | 440 |
| npMoveTo    | 415 |
oldHistFileName, 819
oldParseString, 1448
om-bindTCP, 1291
om-closeConn, 1290
om-closeDev, 1290
om-connectTCP, 1292
om-getApp, 1293
om-getAtP, 1294
om-getAttr, 1294
om-getBind, 1294
om-getBVar, 1294
om-getByteArray, 1295
om-getConnInDev, 1291
om-getConnOutDev, 1291
om-getEndApp, 1295
om-getEndAtP, 1295
om-getEndAttr, 1295
om-getEndBind, 1296
om-getEndBVar, 1296
om-getEndError, 1296
om-getEndObject, 1296
om-getError, 1297
om-getFloat, 1297
om-getInt, 1297
om-getObject, 1297
om-getString, 1298
om-getSymbol, 1298
om-getType, 1298
om-getVar, 1298
om-listCDs, 1288
om-listSymbols, 1288
om-makeConn, 1290
om-openFileDev, 1289
om-openStringDev, 1290
om-putApp, 1299
om-putAtP, 1299
om-putAttr, 1299
om-putBind, 1299
om-putBVar, 1300
om-putByteArray, 1300
om-putEndApp, 1300
om-putEndAtP, 1300
om-putEndAttr, 1301
om-putEndBind, 1301
om-putEndBVar, 1301
om-putEndError, 1301
om-putEndObject, 1302
om-putError, 1302
om-putFloat, 1302
om-putInt, 1302
om-putObject, 1303
om-putString, 1303
om-putSymbol, 1303
om-putVar, 1303
om-Read, 1287
om-setDevEncoding, 1289
om-stringPtrToString, 1304
om-stringToStringPtr, 1304
om-supportsCD, 1288
om-supportsSymbol, 1288
openOutputLibrary, 905
openServer, 1133
operationOpen, 1153
optionError, 721
optionUserLevelError, 722
opTran, 553
orderBySlotNumber, 1068
originsInOrder, 1540
parseAndEval, 1447
parseAndEval1, 1447
parseAndInterpret, 272
parseFromString, 273
parseNoMacroFromString, 1531
parseSystemCmd, 740
parseWord, 1463
pathname, 1192
pathnameDirectory, 1192
pathnameName, 1191
pathnameType, 1191
pathnameTypeLd, 1191
patternVarsOf, 551
patternVarsOf1, 551
pcounters, 1053
permuteToOrder, 687
pf0ApplicationArgs, 467
pf0AssignLhsItems, 486
pf0DefinitionLhsItems, 492
pf0FlattenSyntacticTuple, 467
pf0ForInLhs, 497
pf0FreeItems, 496
pf0LambdaArgs, 504
pf0LocalItems, 505
pfSexpr, strip, 480
pfSourcePosition, 468
pfSourceStok, 473
pfSpread, 469
pfSuch, 474
pfSuchthat, 518
pfSuchThat2Sex, 537
pfSuchthat?, 518
pfSuchthatCond, 518
pfSymb, 481
pfSymbol, 481
pfSymbol?, 482
pfSymbolSymbol, 482
pfTagged, 518
pfTagged?, 519
pfTaggedExpr, 519
pfTaggedTag, 519
pfTaggedToTyped, 519
pfTaggedToTyped1, 474
pfTransformArg, 474
pfTree, 482
pfTuple, 522
pfTuple?, 522
pfTupleListOf, 522
pfTupleParts, 523
pfTweakIf, 520
pfTyped, 520
pfTyped?, 521
pfTypedId, 521
pfTypedType, 521
pfTyping, 521
pfUnSequence, 523
pfWDec, 523
pfWDeclare, 524
pfWhere, 524
pfWhere?, 524
pfWhereContext, 525
pfWhereExpr, 525
pfWhile, 525
pfWhile?, 526
pfWhileCond, 526
pfWith, 526
pfWrong, 526
pfWrong?, 527
phInterpret, 292
phIntReportMsgs, 291
phMacro, 449
phParse, 291
pileCforest, 562
pileColumn, 559
pileCtree, 562
pileForest, 560
pileForest1, 560
pileForests, 559
pilePlusComment, 558
pilePlusComments, 558
pileTree, 559
placep, 1227
pmDontQuote?, 539
pname, 1195
poCharPosn, 599
poFileName, 582
poGetLineObject, 583
poGlobalLinePosn, 296
poLinePosn, 583
poNoPos?, 582
poNoPosition, 636
poNoPosition?, 635
poPosImmediate?, 582
porigin, 313
posend, 352
posPointers, 600
ppos, 579
pquit, 870
pquitSpad2Cmd, 870
previousInterpreterFrame, 18
printAsTeX, 285
printLabelledList, 746
printStatisticsSummary, 283
printStorage, 283
printSynonyms, 745
printTypeAndTime, 284
probeName, 1130
processChPosesForOneLine, 598
processInteractive, 274
processInteractive1, 277
processKeyedError, 575
processMsgList, 591
processSynonymLine, 1039
processSynonymLine, removeKeyFromLine, 1039
processSynonyms, 257
INDEX

protectedEVAL, 271
prTraceNames, 80
prTraceNames, fn, 79
pSearch, 1499
pspacers, 1052
ptimers, 1052
punctuation?, 341
put, 1189
putDatabaseStuff, 587
putFTText, 601
putHist, 832
pvarCondList, 1434
pvarCondList1, 1434
pvarPredTran, 552
pvarsOfPattern, 1435
qenum, 1236
qsquotient, 1243
qsremainder, 1243
queryClients, 777
queueUpErrors, 594
quit, 872
quitSpad2Cmd, 872
quoteString, 1449
quotient2, 1239
random, 1239
rassocSub, 1058
rdefinistream, 1240
rdedefoutstream, 1240
rdigit?, 356
read, 875
readHiFi, 843
readline, 1565
readSpad2Cmd, 875
readSpadProfileIfThere, 1107
reassembleTowerIntoType, 689
reclaim, 263
recordAndPrint, 282
recordFrame, 1083
recordNewValue, 833
recordNewValue0, 833
recordOldValue, 833
recordOldValue0, 834
reduceAlistForDomain, 1519
redundant, 596
regress, 882
remainder2, 1239
remFile, 579
remLine, 584
removeOption, 1054
remover, 1070
removeTracedMapSigs, 61
removeUndoLines, 41
renamePatternVariables, 1431
renamePatternVariables1, 1431
rep, 593
replaceFile, 1131
replacePercentByDollar, 1410
replacePercentByDollar, fn, 1409
replaceSharps, 1104
reportInstantiations, 1205
reportOperations, 1021
reportOpsFromLisplib, 1024
reportOpsFromLisplib0, 1023
reportOpsFromLisplib1, 1024
reportOpsFromUnitDirectly, 1028
reportOpsFromUnitDirectly0, 1027
reportOpsFromUnitDirectly1, 1032
reportSpadTrace, 76
reportUndo, 43
reportWhatOptions, 1093
reroot, 264
resetCounters, 1051
resetHashtables, 1146
resetInCoreHist, 830
resetSpacers, 1051
resetStackLimits, 244
resetTimers, 1051
resetWorkspaceVariables, 894
restart0, 241
restoreHistory, 839
resultp, 888
retract, 1203
retract2Specialization, 704
retractByFunction, 710
retractUnderDomain, 709
rread, 847
ruleLhsTran, 552
rulePredicateTran, 549
runspad, 243
rwrite, 846
safeWritify, 849
sameMsg?, 598
sameUnionBranch, 285
satisfiesRegularExpressions, 1095
satisfiesUserLevel, 723
saveDependentsHashTable, 1168
saveHistory, 837
saveMapSig, 53
savesystem, 891
saveUsersHashTable, 1168
sayAllCacheCounts, 914
sayBrightly1, 1200
sayCacheCount, 915
sayExample, 800
sayKeyedMsg, 27
sayKeyedMsgLocal, 28
sayMSG, 29
sayMSG2File, 28
sayShowWarning, 1032
scanCheckRadix, 357
scanCloser?, 349
scanComment, 339
scanDictCons, 360
scanError, 358
scanEsc, 346
scanEscape, 358
scanExponent, 349
scanIgnoreLine, 336
scanInsert, 361
scanKeyTableCons, 359
scanKeyTr, 343
scanNegComment, 340
scanNumber, 355
ScanOrPairVec, 858
ScanOrPairVec, ScanOrInner, 857
scanPossFloat, 344
scanPunCons, 362
scanPunct, 341
scanS, 354
scanSpace, 352
scanString, 353
scanToken, 337
scanTransform, 355
scanW, 351
scanWord, 349
search, 1110
searchCurrentEnv, 1110
searchTailEnv, 1111
sec, 1221
sech, 1223
segmentKeyedMsg, 28
selectOption, 751
selectOptionLC, 751
separatePiles, 563
serverReadLine, 268
set, 1013
set-restart-hook, 237
set1, 1014
setCurrentLine, 266
setDatabase, 1155
setExposure, 90
setExposureAdd, 91
setExposureAddConstr, 92
setExposureAddGroup, 89
setExposureDrop, 93
setExposureDropConstr, 96
setExposureDropGroup, 94
setFortDir, 930
setFortPers, 961
setFortTempDir, 928
setFunctionsCache, 911
setHistoryCore, 826
setInputLibrary, 906
setIOIndex, 841
setLinkerArgs, 933
setMsgCatlessAttr, 587
setMsgForcedAttr, 586
setMsgForcedAttrList, 586
setMsgUnforgottenAttr, 589
setMsgUnforgottenAttrList, 588
setNagHost, 959
setOutputAlgebra, 967
setOutputCharacters, 971
setOutputFormula, 995
setOutputFortran, 974
setOutputHtml, 980
setOutputLibrary, 904
setOutputMathml, 986
setOutputOpenMath, 990
setOutputTex, 1001
setStreamsCalculate, 1007
setUpDefault, 1428
shortenForPrinting, 1067
show, 1019
null
INDEX

*browse-stream*, 1145
*browse-stream-stamp*, 1145
*category-stream*, 1145
*category-stream-stamp*, 1146
*defaultdomain-list*, 1142
*eof*, 246
*hasCategory-hash*, 1143
*interp-stream*, 1144
*interp-stream-stamp*, 1144
*miss*, 1143
*monitor-domains*, 1313
*monitor-nrlibs*, 1313
*monitor-table*, 1314
*ok*, 884
*operation-hash*, 1143
*operation-stream*, 1144
*operation-stream-stamp*, 1145
*read-place-holder*, 1227
*sourcefiles*, 1164
*whitespace*, 246
/editfile, 781
$NagMessages, 962
$reportBottomUpFlag, 951
$AnonymousFunction, 640
$Any, 640
$BFtag, 640
$BigFloat, 642
$Boolean, 641
$BreakMode, 901
$Category, 641
$CommandSynonymAlist, 749
$ComplexInteger, 643
$Domain, 641
$DoubleFloat, 642, 647
$EmptyMode, 640
$EndServerSession, 268
$Exit, 641
$Expression, 641
$Float, 642
$FontTable, 643
$FunctionalExpression, 646
$HTCompanionWindowID, 277
$HiFiAccess, 31, 937
$HistListAct, 32
$HistListLen, 32
$HistList, 31
$HistRecord, 32
$IOindex, 24
$InitialCommandSynonymAlist, 748
$InitialModemapFrame, 130
$Integer, 643
$InteractiveFrame, 23
$InteropMode, 247
$Mode, 643
$NeedToSignalSessionManager, 268
$NegativeInteger, 643
$NonNegativeInteger, 644
$NonNullStream, 853
$NonPositiveInteger, 644
$NullStream, 854
$OutputForm, 642
$PositiveInteger, 644
$PrintCompilerMessageIfTrue, 242
$ProcessInteractiveValue, 276
$QuietCommand, 271
$QuotientField, 645
$RTspecialCharacters, 1124
$RationalNumber, 644
$SingleFloat, 646
$SingleInteger, 647
$SmallInteger, 646
$SpadServerName, 133
$SpadServer, 133
$StringCategory, 645
$String, 645
$Symbol, 645
$ThrowAwayMode, 280
$UserLevel, 1012
$Void, 645
$abbreviateTypes, 965
$activePageList, 1403
$algebraFormat, 966
$algebraOutputFile, 966
$algebraOutputStream, 967
$attrCats, 586
$bcParseOnly, 1403
$boot, 248
$cacheAlist, 911
$clearAlist, 767
$clearOptions, 766
$compileRecurrence, 917
$compilingMap, 274
$constructors, 1070
$curpage, 1403
$current-directory, 129, 266
$currentFrameNum, 268
$defaultFortranType, 921
$defaultFunctionTargets, 646
$defaultSpecialCharacters, 1121
$describeOptions, 789
$directory-list, 130
$displayDroppedMap, 943
$displayMsgNumber, 950
$displayOptions, 795
$displaySetValue, 952
$displayStartMsgs, 953
$domPvar, 274
$envHashTable, 1109
$env, 247
$e, 247
$formulaFormat, 994
$formulaOutputFile, 994
$fortIndent, 919
$fortInts2Floats, 918
$fortLength, 920
$fortPersistence, 960
$fortranArrayStartingIndex, 926
$fortranDirectory, 930
$fortranFormat, 973
$fortranLibraries, 932
$fortranOptimizationLevel, 925
$fortranOutputFile, 973
$fortranPrecision, 922
$fortranSegment, 925
$fortranTmpDir, 927
$fractionalDisplayType, 978
$frameAlist, 267
$frameMessages, 945
$frameNumber, 267
$frameRecord, 35
$fullScreenSysVars, 934
$functionTable, 770
$funnyBacks, 1450
$funnyQuote, 1449
$genValue, 278
$giveExposureWarning, 943
$globalExposureGroupAlist, 99
$highlightAllowed, 946
$historyDirectory, 819
$historyDisplayWidth, 935, 1475
$historyFileType, 33, 818
$htLineList, 1403
$htMacroTable, 1331
$htmlFormat, 979
$htmlOutputFile, 979
$imPrGuys, 580
$imPrTagGuys, 589
$inputPromptType, 951
$instantRecord, 274
$intCoerceFailure, 255
$intRestart, 249
$intSpadReader, 255
$intTopLevel, 249
$internalHistoryTable, 32
$interpOnly, 273
$interpreterFrameName, 23
$interpreterFrameRing, 23
$lambdaType, 902
$library-directory-list, 130
$linearFormatScripts, 999
$linelength, 983
$localExposureDataDefault, 99
$localExposureData, 98
$margin, 983
$mathmlFormat, 985
$mathmlOutputFile, 985
$maximumFortranExpressionLength, 924
$minivectorNames, 273
$mkTestFlag, 281
$msgAlist, 26
$msgDatabaseName, 131
$msgdbNoBlanksAfterGroup, 27
$msgdbNoBlanksBeforeGroup, 27
$msgdbPrims, 26
$msgdbPunct, 27
$nagEnforceDouble, 963
$nagHost, 958
$nagMessages, 949
$ncMsgList, 250
$newcompErrorCount, 251
$newline, 1475
$newspad, 248
$noEvalTypeMsg, 1082
$noParseCommands, 716
INDEX

$noSubsumption, 1108
$nopos, 251
$npTokToNames, 429
$numberOfEquations, 1380
$numericFailure, 1242
$oldBreakMode, 1242
$oldHistoryFileName, 818
$oldline, 721
$openMathFormat, 989
$openMathOutputFile, 990
$openServerIfTrue, 131
$optionAlist, 1050
$pfMacros, 324
$plainRTspecialCharacters, 1124
$plainSpecialCharacters0, 1122
$plainSpecialCharacters1, 1122
$plainSpecialCharacters2, 1123
$plainSpecialCharacters3, 1123
$preLength, 577
$prettyprint, 1011
$previousBindings, 36
$primitiveHtCommands, 1331
/printAnyIfTrue, 939
/printFortranDecs, 921
/printLoadMsgs, 940
/printMsgsToFile, 944
/printStatisticsSummaryIfTrue, 954
/printTimeIfTrue, 955
/printTypeIfTrue, 956
/printVoidIfTrue, 957
$quitCommandType, 1005
$quitTag, 243
$relative-directory-list, 132
$relative-library-directory-list, 132
$repGuys, 597
$reportBottomUpFlag, 941
$reportCoerceIfTrue, 942
$reportCompilation, 1010
$reportInstantiations, 947
$reportInterpOnly, 948
$reportOptimization, 1010
$reportSpadtrace, 1050
$reportundo, 36
$runTestFlag, 281
$setOptionNames, 1013
$sockBufferLength, 268
$solutionMethod, 1380
$spad-errors, 1113
$spadroot, 133
$specialCharacterAlist, 1125
$specialCharacters, 1125
$streamCount, 1006
$streamsShowAll, 1008
$syscommands, 716
$systemCommands, 715
$systemType, 1379
$testingErrorPrefix, 26
$testingSystem, 955
$texFormat, 1001
$texOutputFile, 1001
$toWhereGuys, 585
$tokenCommands, 747
$traceNoisely, 1049
$traceOptionList, 1050
$tracedMapSignatures, 1050
$underbar, 828
$useEditorForShowOutput, 999
$useFullScreenHelp, 936
$useInternalHistoryTable, 819
$useIntrinsicFunctions, 923
$whatOptions, 1090
boot-line-stack, 1107
creditlist, 127
curinstream, 245
curoutstream, 245
dalymode, 908
dbDelimiters, 1495
echo-meta, 1108
ElseifKeepPart, 303
ElseifSkipPart, 303
ElseifSkipToEnd, 303
ElseKeepPart, 304
ElseSkipToEnd, 303
errorinstream, 245
erroroutstream, 245
file-closed, 1108
IfKeepPart, 302
IfSkipPart, 302
IfSkipToEnd, 302
in-stream, 1107
incCommands, 324
infgeneric, 333
input-libraries, 907
delasc
called by spadUntrace, 78
Delay, 329
called by incAppend, 312
called by incIgen, 299
called by incLude, 301
called by incRgen, 329
called by incZip, 299
called by next, 262
defun, 329
deldatabase, 1155
called by abbreviationsSpad2Cmd, 755
defun, 1155
delete
called by breaklet, 86
called by domainDescendantsOf, 1507
called by setExposeAddConstr, 93
called by setExposeDropConstr, 96
called by setExposeDropGroup, 95
called by trace1, 48
called by tracelet, 85
called by untraceDomainConstructor, 73
called by workfilesSpad2Cmd, 1099
deleteAssoc
called by clearCmdParts, 773
deleteFile, 1193
called by clearCmdAll, 771
calls erase, 1193
calls pathname, 1193
uses $erase, 1193
defun, 1193
defmacro, 1227
descendantsOf
called by kcdPage, 1520
describe, 789, 790
calls describespad2cmd, 790
defun, 790
manpage, 789
describeFortPersistence, 961
called by setFortPers, 961
calls sayBrightly, 961
uses $fortPersistence, 961
defun, 961
describeInputLibraryArgs, 907
called by setInputLibrary, 906
calls sayBrightly, 907
defun, 907
describeOutputLibraryArgs, 904
called by setOutputLibrary, 904
calls sayBrightly, 904
defun, 904
describeSetFortDir, 931
called by setFortDir, 930
calls sayBrightly, 931
uses $fortranDirectory, 931
defun, 931
describeSetFortTmpDir, 929
called by setFortTmpDir, 928
calls sayBrightly, 929
uses $fortranTmpDirectory, 929
defun, 929
describeSetFunctionsCache, 914
called by setFunctionsCache, 911
calls sayBrightly, 914
defun, 914
describeSetLinkerArgs, 933
called by setLinkerArgs, 932
calls sayBrightly, 933
uses $fortranLibraries, 933
defun, 933
describeSetNagHost, 959
called by setNagHost, 959
calls sayBrightly, 959
uses $nagHost, 959
defun, 959
describeSetOutputAlgebra, 969
called by setOutputAlgebra, 968
INDEX

calls sayBrightly, 969
calls setOutputAlgebra, 969
defun, 969
describeSetOutputFormula, 997
called by setOutputFormula, 995
calls sayBrightly, 997
calls setOutputFormula, 998
defun, 997
describeSetOutputFortran, 977
called by setOutputFortran, 974
calls sayBrightly, 977
calls setOutputFortran, 977
defun, 977
describeSetOutputHtml, 982
called by setOutputHtml, 980
calls sayBrightly, 982
calls setOutputHtml, 982
defun, 982
describeSetOutputMathml, 988
called by setOutputMathml, 986
calls sayBrightly, 988
calls setOutputMathml, 988
defun, 988
describeSetOutputOpenMath, 993
called by setOutputOpenMath, 990
calls sayBrightly, 993
calls setOutputOpenMath, 993
defun, 993
describeSetOutputTex, 1004
called by setOutputTex, 1002
calls sayBrightly, 1004
calls setOutputTex, 1004
defun, 1004
describeSetStreamsCalculate, 1007
called by setStreamsCalculate, 1007
calls sayKeyedMsg, 1007
tuses $streamCount, 1008
defun, 1007
describeSpad2Cmd, 790
calls cleanline, 790
calls flatten, 790
calls getdatabase, 790
calls sayMessage, 790
calls selectOptionLC, 790
tuses $EmptyEnvironment, 790
tuses $describeOptions, 790

uses $e, 790
defun, 790
describespad2cmd
called by describe, 790
desiredMsg, 575
called by ncHardError, 574
called by ncSoftError, 574
defun, 575
devulate
called by /tracereply, 77
called by ?t, 84
called by addTraceItem, 83
called by dbAddChainDomain, 1544
called by dbSearchOrder, 1514
called by mkEvalable, 1076
called by prTraceNames, fn, 80
called by shortenForPrinting, 1067
called by spadReply, printName, 1069
called by spadUntrace, 78
called by trace1, 48
called by transTraceItem, 60
called by untraceDomainConstructor, keepTraced?, 73
called by writify, writifyInner, 849
dewritify, 857
called by SPADRREAD, 847
calls ScanOrPairVec, 857
calls dewritify, dewritifyInner, 857
calls function, 857
tuses $seen, 857
defun, 857
dewritify, dewritifyInner, 854
called by dewritify, dewritifyInner, 854
called by dewritify, 857
calls concat, 854
calls dewritify, dewritifyInner, 854
calls error, 854
calls exit, 854
calls gensymmer, 854
calls hget, 854
calls hput, 854
calls intp, 854
calls make-instream, 854
calls poundsign, 854
calls qcar, 854
calls qcdr, 854
calls qrplaca, 854
    calls qrplacd, 854
    calls qsetvelt, 854
    calls qvelt, 854
    calls qvmaxindex, 854
    calls seq, 854
    calls vecp, 854
    calls vmread, 854
    uses $NonNullStream, 854
    uses $NullStream, 854
    uses $seen, 854
    defun, 854
    DFAcos, 1216
      defmacro, 1216
    DFAcosh, 1219
      defmacro, 1219
    DFAdd, 1212
      defmacro, 1212
    DFAsin, 1216
      defmacro, 1216
    DFAsinh, 1218
      defmacro, 1218
    DFTan, 1217
      defmacro, 1217
    DFTanh, 1218
      defmacro, 1218
    DFUnaryMinus, 1211
      defmacro, 1211
    DFZerop, 1211
      defmacro, 1211
diffAlist, 1085
      calledby recordFrame, 1083
calls assoc, 1085
calls assq, 1085
calls exit, 1085
calls lassq, 1085
calls reportUndo, 1085
calls seq, 1085
calls tmp1, 1085
defun, 1085
dig2fix
      calledby isSharpVarWithNum, 1063
digit?, 345
      calledby scanExponent, 350
calls digitp, 345
defun, 345
digitp, 1195
called by digit?, 345
called by digitp, 1195
called by isSharpVarWithNum, 1063
called by isgenvar, 1064
calls digitp, 1195
defun, 1195

DirToString, 1229
called by fnameDirectory, 1229
defun, 1229
disableHist, 845
called by fetchOutput, 842
called by historySpa2Cmd, 822
called by restoreHistory, 839
called by setHistoryCore, 826
called by showInOut, 842
called by showInput, 841
called by undoFromFile, 836
called by undoInCore, 834
called by updateHist, 831
calls histFileErase, 845
calls histFileName, 845
uses $HiFiAccess, 845
defun, 845
display, 794, 796
calls displaySpa2Cmd, 796
defun, 796
manpage, 794
displayCondition, 736
called by displayProperties, 733
calls bright, 736
calls concat, 736
calls pred2English, 736
calls sayBrightly, 736
defun, 736
displayDomainOp
called by dbShowConsDoc1, 1550
displayExposedConstructors, 97
called by setExposeAddConstr, 93
called by setExposeAddGroup, 89
called by setExposeAdd, 91
called by setExposeDropConstr, 96
called by setExposeDropGroup, 94
called by setExpose, 90
calls sayKeyedMsg, 97
uses $localExposureData, 97
defun, 97
displayExposedGroups, 97
called by setExposeAddGroup, 89
called by setExposeAdd, 91
called by setExposeDropGroup, 94
called by setExpose, 90
calls sayKeyedMsg, 97
uses $interpreterFrameName, 97
uses $localExposureData, 97
defun, 97
displayFrameNames, 13
called by frameSpa2Cmd, 11
calls bright, 13
calls framename, 13
uses $interpreterFrameRing, 13
defun, 13
displayHiddenConstructors, 98
called by setExposeAddGroup, 89
called by setExposeDropConstr, 96
called by setExposeDropGroup, 95
called by setExposeDrop, 93
called by setExpose, 90
calls sayKeyedMsg, 98
uses $localExposureData, 98
defun, 98
displayMacro, 726
called by displayMacros, 799
called by displayProperties, 733
calls bright, 726
calls isInterpMacro, 726
calls mathprint, 726
calls object2String, 726
calls sayBrightly, 726
calls strconc, 726
uses $op, 726
defun, 726
displayMacros, 799
called by displaySpa2Cmd, 796
calls displayMacro, 799
calls displayParserMacro, 799
calls exit, 799
calls getInterpMacroNames, 799
calls getParserMacroNames, 799
calls member, 799
calls remdup, 799
calls sayBrightly, 799
calls \texttt{seq}, 799
defun, 799
displayMode, 738
calledby displayProperties, 733
calls bright, 738
calls concat, 738
calls \texttt{fixObjectForPrinting}, 738
calls \texttt{prefix2String}, 738
calls \texttt{sayBrightly}, 738
defun, 738
displayModemap, 738
calledby displayProperties, 733
calls bright, 738
calls concat, 738
calls formatSignature, 738
calls \texttt{sayBrightly}, 738
defun, 738
displayOperations, 798
calledby displaySpad2Cmd, 796
calls reportOpSymbol, 798
calls \texttt{sayKeyedMsg}, 798
calls yesanswer, 798
defun, 798
displayOperationsFromLisplib, 1027
calledby reportOpsFromLisplib, 1025
calls eq substlist, 1027
calls formatOperationAlistEntry, 1027
calls getdatabase, 1027
calls msort, 1027
calls remdup, 1027
calls reportOpsFromUnitDirectly, 1027
calls \texttt{say2PerLine}, 1027
calls specialChar, 1027
calls \texttt{qcar}, 1027
calls \texttt{qcdr}, 1027
calls \texttt{remdup}, 1027
calls getalist, 1027
calls \texttt{say MSG}, 1027
calls \texttt{seq}, 1027
calls \texttt{terminateSystemCommand}, 1027
calls \$\texttt{dependeeAlist}, 1027
calls \$\texttt{dependentAlist}, 1027
defun, 1027
displayParserMacro, 736
calledby displayMacros, 799
calledby displayProperties, 733
calls pfPrintSrcLines, 736
calls \$\texttt{pf Macros}, 736
defun, 736
displayProperties, 733
calledby displaySpad2Cmd, 796
calls bright, 733
calls displayCondition, 733
calls displayMacro, 733
calls displayModemap, 733
calls displayMode, 733
calls displayParserMacro, 733
calls displayProperties, sayFunctionDepes, 733
calls displayType, 733
calls displayValue, 733
calls exit, 733
calls \texttt{fixObjectForPrinting}, 733
calls getAndSay, 733
calls getIPropList, 733
calls getInterpMacroNames, 733
calls getI, 733
calls getParserMacroNames, 733
calls getWorkspaceNames, 733
calls interpFunctionDepAlists, 733
calls isInternalMapName, 733
calls isInterpMacro, 733
calls member, 733
calls msort, 733
calls prefix2String, 733
calls \texttt{qcar}, 733
calls \texttt{qcdr}, 733
calls remdup, 733
calls sayKeyedMsg, 733
calls sayMSG, 733
calls \texttt{seq}, 733
calls \texttt{terminateSystemCommand}, 733
calls \$\texttt{dependeeAlist}, 733
calls \$\texttt{dependentAlist}, 733
calls \$\texttt{frameMessages}, 733
calls \$\texttt{interpreterFrameName}, 733
defun, 733
displayProperties, sayFunctionDepes, 728
calledby displayProperties, 733
calls bright, 730
calls exit, 730
calls getalist, 730
calls \texttt{say MSG}, 730
calls \texttt{seq}, 730
calls \$\texttt{dependeeAlist}, 730
calls \$\texttt{dependentAlist}, 730
defun, 728
displayRule
calledby displayValue, 731
INDEX

displaySetOptionInformation, 896
  called by set1, 1014
calls boot-equal, 896
calls bright, 896
calls concat, 896
calls displaySetVariableSettings, 896
calls eval, 896
calls literals, 896
calls object2String, 896
calls sayBrightly, 896
calls sayMSG, 896
calls sayMessage, 896
calls specialChar, 896
calls translateTrueFalse2YesNo, 896
  uses $linelength, 896
defun, 896

displaySetVariableSettings, 897
  called by displaySetOptionInformation, 896
called by set1, 1014
calls bright, 897
calls concat, 897
calls eval, 897
calls fillerSpaces, 897
calls literals, 897
calls object2String, 897
calls poundsign, 897
calls satisfiesUserLevel, 897
calls sayBrightly, 897
calls say, 897
calls specialChar, 897
calls translateTrueFalse2YesNo, 897
  calls tree, 897
  uses $linelength, 897
defun, 897

displaySpad2Cmd
  calls abbrevQuery, 796
calls displayMacros, 796
calls displayOperations, 796
calls displayProperties, 796
calls displayWorkspaceNames, 796
calls listConstructorAbbreviations, 796
calls opOf, 796
calls sayMessage, 796
calls selectOptionLC, 796
  uses $EmptyEnvironment, 796
  uses $displayOptions, 796
defun, 796

displayValue, 731
  called by displayProperties, 733
calls concat, 731
calls displayRule, 731
calls fixObjectForPrinting, 731
calls form2String, 731
calls getdatabase, 731
calls mathprint, 731
calls objMode, 731
calls objValUnwrap, 731
calls outputFormat, 731
calls pfname, 731
calls prefix2String, 731
calls sayMSG, 731
calls stronc, 731
  uses $EmptyMode, 731
  uses $op, 731
defun, 731

displayWorkspaceNames, 726
  called by displaySpad2Cmd, 796
calls getInterpMacroNames, 726
calls getParserMacroNames, 726
calls getWorkspaceNames, 727
calls msort, 727
calls sayAsManyPerLineAsPossible, 727
calls sayBrightly, 727
calls sayMessage, 726
calls setdifference, 727
defun, 726
den, 1226
defmacro, 1226
doDoitButton, 1442
defun, 1442
domainDescendantsOf, 1507
called by kArgPage, 1506
calls assoc, 1507
calls delete, 1507
calls domainsOf, 1507
calls function, 1507
calls ifcdr, 1507
calls listSort, 1507
calls qcar, 1507
calls qcdr, 1507
calls quickAnd, 1507
calls simpHasPred, 1507
calls systemError, 1507
defun, 1507
domainsOf
called by domainDescendantsOf, 1507
called by kcdolPage, 1521
called by make-databases, 1165
domainToGenvar, 1054
called by getTraceOption,hn, 54
called by transTraceItem, 60
calls evalDomain, 1054
calls genDomainTraceName, 1054
calls getdatabase, 1054
calls opOf, 1054
calls unabbrevAndLoad, 1054
uses $doNotAddEmptyModeIfTrue, 1054
defun, 1054
domArg, 652
called by hasCaty, 650
called by mkDomPvar, 663
local ref $FormalMapVariableList, 652
defun, 652
domArg2, 652
called by hasCaty, 650
calls isSharpVar, 652
calls subCopy, 652
local ref $domPvar, 652
defun, 652
done
called by insertPos, 601
doSystemCommand, 718
called by ExecuteInterpSystemCommand, 256
calls concat, 718
calls expand-tabs, 718
calls getFirstWord, 718
calls handleNoParseCommands, 718
calls handleParsedSystemCommands, 718
calls handleTokenizeSystemCommands, 718
calls member, 718
calls processSynonyms, 718
calls splitIntoOptionBlocks, 718
calls substring, 718
calls unAbbreviateKeyword, 718
uses $noParseCommands, 718
uses $tokenCommands, 718
uses line, 718
defun, 718
downcase, 1206
called by apropos, 1097
called by conPage, 1503
called by constructorSearch, 1501
called by dbShowCons, 1545
called by koaPageFilterByName, 1538
called by selectOptionLC, 751
called by set1, 1014
called by setOutputCharacters, 971
called by string2Constructor, 1494
called by whatSpad2Cmd, 1091
calls identp, 1206
defun, 1206
downlink, 1475
called by conPage, 1503
called by constructorSearch, 1501
defun, 1475
dqAppend, 566
called by dqConcat, 565
called by pileCtree, 562
defun, 566
dqConcat, 565
called by dqConcat, 565
called by enPile, 562
called by separatePiles, 563
calls dqAppend, 565
calls dqConcat, 565
defun, 565
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dqToList</td>
<td>566</td>
</tr>
<tr>
<td></td>
<td>called by intloopEchoParse, 294</td>
</tr>
<tr>
<td></td>
<td>called by ncloopParse, 262</td>
</tr>
<tr>
<td></td>
<td>defun, 566</td>
</tr>
<tr>
<td>dqUnit</td>
<td>565</td>
</tr>
<tr>
<td></td>
<td>called by enPile, 562</td>
</tr>
<tr>
<td></td>
<td>called by lineoftoks, 335</td>
</tr>
<tr>
<td></td>
<td>called by scanToken, 338</td>
</tr>
<tr>
<td></td>
<td>called by separatePiles, 563</td>
</tr>
<tr>
<td></td>
<td>defun, 565</td>
</tr>
<tr>
<td>drop[9]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>called by frameSpad2Cmd, 11</td>
</tr>
<tr>
<td>dropInputLibrary</td>
<td>908</td>
</tr>
<tr>
<td></td>
<td>called by addInputLibrary, 907</td>
</tr>
<tr>
<td></td>
<td>called by openOutputLibrary, 905</td>
</tr>
<tr>
<td></td>
<td>called by setInputLibrary, 906</td>
</tr>
<tr>
<td></td>
<td>uses input-libraries, 908</td>
</tr>
<tr>
<td></td>
<td>defun, 908</td>
</tr>
<tr>
<td>dropLeadingBlanks</td>
<td></td>
</tr>
<tr>
<td></td>
<td>called by processSynonymLine, removeKeyFromLine, 1039</td>
</tr>
<tr>
<td>dSearch</td>
<td>1499</td>
</tr>
<tr>
<td></td>
<td>calls constructorSearch, 1499</td>
</tr>
<tr>
<td></td>
<td>defun, 1499</td>
</tr>
<tr>
<td>dsetaref2</td>
<td>1225</td>
</tr>
<tr>
<td></td>
<td>defmacro, 1225</td>
</tr>
<tr>
<td>dsetelt</td>
<td>1227</td>
</tr>
<tr>
<td></td>
<td>defmacro, 1227</td>
</tr>
<tr>
<td>dumbTokenize</td>
<td>739</td>
</tr>
<tr>
<td></td>
<td>called by handleParsedSystemCommands, 740</td>
</tr>
<tr>
<td></td>
<td>called by handleTokensizeSystemCommand, 719</td>
</tr>
<tr>
<td></td>
<td>called by parseSystemCmd, 740</td>
</tr>
<tr>
<td></td>
<td>uses stripSpaces, 739</td>
</tr>
<tr>
<td></td>
<td>defun, 739</td>
</tr>
<tr>
<td>echo-meta</td>
<td>1108</td>
</tr>
<tr>
<td></td>
<td>defvar, 1108</td>
</tr>
<tr>
<td>edit</td>
<td>803</td>
</tr>
<tr>
<td></td>
<td>804</td>
</tr>
<tr>
<td></td>
<td>called by editSpad2Cmd, 804</td>
</tr>
<tr>
<td></td>
<td>defun, 804</td>
</tr>
<tr>
<td></td>
<td>manpage, 803</td>
</tr>
<tr>
<td>editFile</td>
<td>805</td>
</tr>
<tr>
<td></td>
<td>called by editSpad2Cmd, 804</td>
</tr>
<tr>
<td></td>
<td>called by reportOpsFromUnitDirectly1, 1024</td>
</tr>
<tr>
<td></td>
<td>called by reportOpsFromUnitDirectly1, 1032</td>
</tr>
<tr>
<td></td>
<td>calls namestring, 805</td>
</tr>
<tr>
<td></td>
<td>calls obey, 805</td>
</tr>
<tr>
<td></td>
<td>calls pathname, 805</td>
</tr>
<tr>
<td></td>
<td>calls strconc, 805</td>
</tr>
<tr>
<td></td>
<td>defun, 805</td>
</tr>
<tr>
<td>editSpad2Cmd</td>
<td>804</td>
</tr>
<tr>
<td></td>
<td>called by edit, 804</td>
</tr>
<tr>
<td></td>
<td>calls $FINDFILE, 804</td>
</tr>
<tr>
<td></td>
<td>calls editFile, 804</td>
</tr>
<tr>
<td></td>
<td>calls pathnameDirectory, 804</td>
</tr>
<tr>
<td></td>
<td>calls pathnameName, 804</td>
</tr>
<tr>
<td></td>
<td>calls pathnameType, 804</td>
</tr>
<tr>
<td></td>
<td>calls pathname, 804</td>
</tr>
<tr>
<td></td>
<td>calls updateSourceFiles, 804</td>
</tr>
<tr>
<td></td>
<td>uses /editfile, 804</td>
</tr>
<tr>
<td></td>
<td>defun, 804</td>
</tr>
<tr>
<td>Else?,</td>
<td>305</td>
</tr>
<tr>
<td></td>
<td>called by xIfSyntax, 322</td>
</tr>
<tr>
<td></td>
<td>calls quotient, 305</td>
</tr>
<tr>
<td></td>
<td>defun, 305</td>
</tr>
<tr>
<td>Elself?,</td>
<td>304</td>
</tr>
<tr>
<td></td>
<td>called by incLude1, 307</td>
</tr>
<tr>
<td></td>
<td>calls quotient, 304</td>
</tr>
<tr>
<td></td>
<td>defun, 304</td>
</tr>
<tr>
<td>ElselfKeepPart</td>
<td>303</td>
</tr>
<tr>
<td></td>
<td>defvar, 303</td>
</tr>
<tr>
<td>ElselfSkipPart</td>
<td>303</td>
</tr>
<tr>
<td></td>
<td>defvar, 303</td>
</tr>
<tr>
<td>ElseifSkipToEnd</td>
<td>303</td>
</tr>
<tr>
<td></td>
<td>defvar, 303</td>
</tr>
<tr>
<td>ElseIfToEnd</td>
<td>303</td>
</tr>
<tr>
<td></td>
<td>defvar, 304</td>
</tr>
<tr>
<td>ElseOfToEnd</td>
<td>303</td>
</tr>
<tr>
<td></td>
<td>defvar, 303</td>
</tr>
<tr>
<td>eltU16</td>
<td>1248</td>
</tr>
<tr>
<td></td>
<td>defmacro, 1248</td>
</tr>
<tr>
<td>eltU32</td>
<td>1249</td>
</tr>
<tr>
<td></td>
<td>defmacro, 1249</td>
</tr>
<tr>
<td>eltU8</td>
<td>1247</td>
</tr>
<tr>
<td></td>
<td>defmacro, 1247</td>
</tr>
<tr>
<td>embed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>called by traceDomainConstructor, 71</td>
</tr>
<tr>
<td></td>
<td>emptyInterpreterFrame, 12</td>
</tr>
<tr>
<td></td>
<td>called by addNewInterpreterFrame, 18</td>
</tr>
</tbody>
</table>
called by initializeInterpreterFrameRing, 12
  uses $HiFiAccess, 12
  uses $HistList.Act, 12
  uses $HistList.Len, 12
  uses $HistList, 12
  uses $HistRecord, 12
  uses $localExposureDataDefault, 13
defun, 12
emptySearchPage
called by dbShowCons, 1545
enable-backtrace
called by break, 87
called by ncBug, 590
endedp, 887
called by split, 886
defun, 887
enFile, 562
called by pileCforest, 562
calls dqConcat, 562
calls dqUnit, 562
calls firstTokPosn, 562
calls lastTokPosn, 562
calls tokConstruct, 562
defun, 562
entryWidth
called by printLabelledList, 746
eof
  used by fin, 808
eofp, 1129
defun, 1129
eqcar
called by StreamNull, 555
called by hasCateSpecialNew, 666
called by hasCateSpecial, 665
called by pfAbSynOp?, 634
called by poNoPosition?, 635
eqpileTree, 561
called by pileForest1, 560
calls npNull, 561
calls pileColumn, 561
calls pileForests, 561
defun, 561
eqsubstlist
called by displayOperationsFromLisplib, 1027
called by reportOpsFromLisplib, 1024
erase
called by deleteFile, 1193
called by reportOpsFromLisplib1, 1024
called by reportOpsFromUnitDirectly1, 1032
called by saveDependentsHashTable, 1168
called by saveUsersHashTable, 1168
erMsgCompare, 592
calls compareaposns, 592
calls getMsgPos, 592
defun, 592
erMsgSep, 592
called by erMsgSort, 591
calls getMsgPos, 592
calls poNoPos?, 592
defun, 592
erMsgSort, 591
called by processMsgList, 591
calls erMsgSep, 591
calls listSort, 591
defun, 591
error
called by basicLookupCheckDefaults, 1188
called by basicLookup, 1186
called by charDigitVal, 859
called by dewritify, dewritifyInner, 854
called by gensymInt, 859
called by myWritable?, 1231
errormsginstream, 245
defvar, 245
errormsgoutstream, 246
defvar, 246
errorPage
  called by kcPage, 1515
called by kcaPage1, 1521
called by kccPage, 1522
called by kcnPage, 1526
called by kcpPage, 1518
called by kePage, 1509
called by kiPage, 1509
called by koPage, 1535
called by ksPage, 1513
eval
called by NRTevalDomain, 1189
called by displaySetOptionInformation, 896
called by displaySetVariableSettings, 897
evalDomain, 1075
called by evalDomain, 1075
evalCategory, 1105
called by evaluateType1, 1080
calls isPartialMode, 1105
calls oCategory, 1105
defun, 1105
evalDomain, 1075
called by NRTevalDomain, 1189
called by algEqual, 696
called by coerceInt1, 675
called by coerceIntByMap, 693
called by domainToGenvar, 1054
called by getConstantFromDomain, 699
called by reportOpsFromUnitDirectly, 1028
called by retractByFunction, 710
calls concat, 1075
calls eval, 1075
calls mkEvalable, 1075
calls prefix2String, 1075
calls sayMSG, 1075
calls startTimingProcess, 1075
calls stopTimingProcess, 1075
local ref $evalDomain, 1075
defun, 1075
evalSharpOne, 708
called by coerceUnion2Branch, 707
defun, 708
evaluateSignature, 1083
called by evaluateType, 1078
calls evaluateType, 1083
defun, 1083
evaluateType, 1078
called by evaluateSignature, 1083
called by evaluateType1, 1080
called by evaluateType, 1079
called by reportOperations, 1021
calls bottomUp, 1078
calls constructor?, 1079
calls evaluateSignature, 1078
calls evaluateType, 1079
calls getValue, 1078
calls isDomainValuedVariable, 1078
calls member, 1078
calls mkAtree, 1078
calls objVal, 1078
calls qcar, 1078
calls qcdr, 1078
calls throwEvalTypeMsg, 1079
local def $expandSegments, 1079
local ref $EmptyMode, 1079
defun, 1078
evaluateType1, 1080
calls bottomUp, 1080
calls categoryForm?, 1080
calls coerceOrRetract, 1080
calls constructor?, 1080
calls evalCategory, 1080
calls evaluateType, 1080
calls getAndEvalConstructorArgument, 1080
calls getConstructorSignature, 1080
calls getdatabase, 1080
calls makeOrdinal, 1081
calls mkAtree, 1080
calls objValUnwrap, 1080
calls putTarget, 1080
calls qcar, 1080
calls qcdr, 1080
calls replaceSharps, 1080
calls throwEvalTypeMsg, 1080
calls throwKeyedMsgCannotCoerceWith-
Value, 1080
local ref $EmptyMode, 1081
local ref $quadSymbol, 1081
defun, 1080
executeInterpreterCommand, 1442
defun, 1442
ExecuteInterpSystemCommand, 256
called by InterpExecuteSpadSystemCom-
mmand, 256
calls doSystemCommand, 256
calls intProcessSynonyms, 256
calls substring, 256
uses $currentLine, 256
defun, 256
executeQuietCommand, 272
called by serverReadLine, 269
calls make-string, 272
calls parseAndInterpret, 272
calls sockGetString, 272
uses $MenuServer, 272
uses $QuietCommand, 272
catches, 272
defun, 272
exit
calledby /tracereply, 77
calledby abbreviationsSpad2Cmd, 755
calledby apropos, 1097
calledby clearCmdParts, 773
calledby coerceSpadArgs2E, 63
calledby dewritify, dewritifyInner, 854
calledby diffAlist, 1085
calledby displayMacros, 799
calledby displayProperties, sayFunctionDeps, 730
calledby displayProperties, 733
calledby flattenOperationAlist, 1061
calledby findLAM, 1060
calledby getAliasIfTracedMapParameter, 74
calledby getBpiNameIfTracedMap, 75
calledby getPreviousMapSubNames, 1056
calledby getTraceOptions, hn, 54
calledby getTraceOptions, 52
calledby getTraceOption, 55
calledby historySpad2Cmd, 822
calledby importFromFrame, 19
calledby isListOfIdentifiersOrStrings, 1056
calledby isListOfIdentifiers, 1055
calledby orderBySlotNumber, 1068
calledby prTraceNames, fn, 80
calledby prTraceNames, 80
calledby recordFrame, 1083
calledby removeUndoLines, 41
calledby reportUndo, 44
calledby runspsd, 243
calledby set1, 1014
calledby spadReply, printName, 1069

calledby spadReply, 1069
calledby spadTrace, isTraceable, 66
calledby spadTrace, 67
calledby spadUntrace, 78
calledby subTypes, 1055
calledby trace1, 48
calledby traceDomainConstructor, 71
calledby traceReply, 81
calledby undoFromFile, 836
calledby undoLocalModemapHack, 40
calledby undoSingleStep, 39
calledby untraceDomainConstructor, keepTraced?, 73

calledby untraceDomainConstructor, 73
calledby whatConstructors, 1097
calledby whatSpad2Cmd, 1091
calledby writify, writifyInner, 849
expand-tabs
calledby doSystemCommand, 718
calledby include1, 307
explainExact
calledby finalExactRequest, 1396
explainLinear, 1395
calledby linearFinalRequest, 1395
calledby systemError, 1395
defun, 1395
extendLocalLibdb
calledby library, 1159
extractFileNameFromPath
calledby kdPageInfo, 1505
extractHasArgs

calledby augmentHasArgs, 1523
fbpip
calledby mkEvalable, 1076
fetchOutput, 842
called assq, 842
called boot-equal, 842
called disableHist, 842
called getI, 842
called readHiFi, 842
called throwKeyedMsg, 842
defun, 842
file-closed, 1108

called by init-boot/spad-reader, 1114
defvar, 1108
filep
called by setOutputLibrary, 904
fillerSpaces, 242
called by displaySetVariableSettings, 897
called by justifyMyType, 286
called by printLabelledList, 746

called by spadStartUpMsgs, 241
called ifc, 242
defun, 242

called by filterAndFormatConstructors, 1096
called by whatSpad2Cmd, 1091
INDEX

calls blankList, 1096
calls filterListOfStringsWithFn, 1096
calls function, 1096
calls pp2Cols, 1096
calls sayMessage, 1096
calls specialChar, 1096
calls whatConstructors, 1096
uses $linelength, 1096
defun, 1096
filterListOfStrings, 1094
called by apropos, 1097
called by whatCommands, 1093
calls satisfiesRegularExpressions, 1094
defun, 1094
filterListOfStringsWithFn, 1095
called by filterAndFormatConstructors, 1096
called by printSynonyms, 745
calls satisfiesRegularExpressions, 1095
defun, 1095
fin, 808
uses eof, 808
defun, 808
manpage, 808
throws, 808
finalExactRequest, 1395
calls bcQueryInteger, 1395
calls explainExact, 1396
calls moreExactSolution, 1396
calls sayBrightly, 1395
defun, 1395
findfile

called by bcUnixTable, 1554
called by readSpad2Cmd, 875
findFrameInRing, 16
called by changeToNamedInterpreterFrame, 17
calls boot-equal, 16
calls frameName, 16
uses $interpreterFrameRing, 16
defun, 16
findFunctionInDomain
called by retractByFunction, 710
findnexttest, 884
called by regress, 882
calls testnumberp, 884
defun, 884
firstTokPosn, 563
called by enPile, 562
calls tokPosn, 563
defun, 563
fixObjectForPrinting, 727
called by clearCmdParts, 738
called by displayMode, 733
called by displayProperties, 733
called by displayType, 732
called by displayValue, 731
calls member, 728
calls object2Identifier, 728
calls pname, 728
calls strconc, 728
uses $msgdbPrims, 728
defun, 727
flatten, 793
called by describeSpad2Cmd, 790
defun, 793
flattenOperationAlist, 1061
called by spadTrace, 67
calls exit, 1061
calls seq, 1061
defun, 1061
float
called by ptimers, 1052
float2Sex, 535
called by pfLiteral2Sex, 534
uses $useBFasDefault, 535
defun, 535
flowSegmentedMsg

called by msgOutputter, 576
called by msgText, 286
called by sayKeyedMsgLocal, 28
called by traceReply, 81
flung
used by intloopSpadProcess, 289
fnameDirectory, 1229
called by myWritable?, 1231
calls DirToString, 1229
defun, 1229
fnameExists?, 1230
called by myWritable?, 1231
defun, 1230
fnameMake, 1228
called by fnameNew, 1231
calls StringToDir, 1228
defun, 1228
fnameName, 1229
defun, 1229
fnameNew, 1231
calls fnameMake, 1231
defun, 1231
calls fnameMake, 1231
fnameReadable?, 1230
defun, 1230
fnameType, 1230
defun, 1230
fnameWritable?, 1230
calls myWriteable?, 1230
defun, 1230
for, 613
syntax, 613
form2HtString
calledby conPage, 1503
calledby dbConsHeading, 1553
calledby dbSpecialDescription, 1555
calledby dbSpecialExports, 1556
calledby dbSpecialOperations, 1556
calledby kcPage, 1515
calledby kcaPage1, 1521
calledby kcpPage, 1526
calledby kcpPage, 1519
calledby kpPage, 1509
called by koPage, 1535
called by ksPage, 1513
form2LispString
called by libConstructorSig, 1563
form2String
called by displayValue, 731
called by kArgumentCheck, 1529
called by reportOpsFromLisplib, 1025
form2StringWithWhere
called by reportOpsFromLisplib, 1024
formatAttribute
called by reportOpsFromLisplib, 1025
called by reportOpsFromUnitDirectly, 1028
framename
called by addNewInterpreterFrame, 18
called by closeInterpreterFrame, 22
called by displayFrameNames, 13
called by importFromFrame, 19
called by importFromFrame, 19
uses $interpreterFrameRing, 13
defun, 13
formatOpType
called by reportOpsFromUnitDirectly, 1028
formatSignature
called by compiledLookupCheck, 769
called by displayModemap, 738
frame, 3, 10
calls frameSpad2Cmd, 10
defun, 10
manpage, 3
frameEnvironment, 16
called by importFromFrame, 19
calls frameInteractive, 16
defun, 16
frameExposureData, 10
defmacro, 10
frameHiFiAccess, 8
defmacro, 8
frameHistList, 8
defmacro, 8
frameHistListAct, 9
defmacro, 9
frameHistListLen, 9
defmacro, 9
frameHistoryTable, 10
defmacro, 10
frameInteractive, 8
called by frameEnvironment, 16
defmacro, 8
frameIOIndex, 8
defmacro, 8
framenames, 13
called by importFromFrame, 19
framename
called by findFrameInRing, 16
defmacro, 7
formatAttribute
called by reportOpsFromLisplib, 1025
called by reportOpsFromUnitDirectly, 1028
framenames, 13
formatOperation
called by reportOpsFromUnitDirectly, 1028
formatOperationAlistEntry
called by displayOperationsFromLisplib, 1027
framenames, 13
frameSpad2Cmd, 10
called by frame, 10
calls addNewInterpreterFrame, 11
calls closeInterpreterFrame, 11
calls displayFrameNames, 11
calls drop[9], 11
calls helpSpad2Cmd, 10
calls importFromFrame, 11
calls import, 11
calls last, 11
calls names, 11
calls new, 11
calls nextInterpreterFrame, 11
calls next, 11
calls object2Identifier, 11
calls previousInterpreterFrame, 11
calls qcar, 11
calls qcdr, 11
calls selectOptionLC, 10
calls throwKeyedMsg, 10
uses $options, 11
defun, 11
From, 602
called by syIgnoredFromTo, 416
defun, 602
FromTo, 602
called by syIgnoredFromTo, 416
defun, 602
function
called by dbShowCons1, 1547
called by dbShowConstructorLines, 1554
defun, 857
called by dewritify, 857
called by domainDescendantsOf, 1507
defun, 1485
called by filterAndFormatConstructors, 1096
called by isSystemDirectory, 1183
called by kcaPage1, 1521
called by parseNoMacroFromString, 1531
called by writify, 852
gensymInt, 859
defun, 11
called by domainToGenvar, 1054
called by genDomainTraceName, 59
called by genvar, 59
called by lassoc, 59
defun, 59
called by charDigitVal, 859
called by error, 859
called by gensymp, 859
called by pname, 859
defun, 859
called by dewritify, dewritifyInner, 854
called by breaklet, 86
called by gensymInt, 859
called by letPrint2, 1064
called by letPrint3, 1066
called by letPrint, 1062
defun, 859
called by spadTrace, isTraceable, 66
called by tracelet, 85
called by genDomainTraceName, 59
called by letPrint, 1062
called by augmentTraceNames, 65
called by clearCmdParts, 773
called by getAliasIfTracedMapParameter, 74
called by getBpiNameIfTracedMap, 75
called by getMapSig, 54
called by getMapSubNames, 65
called by getPreviousMapSubNames, 1056
called by importFromFrame, 19
called by isDomainValuedVariable, 1105
defun, 1060
called by putHist, 832
defun, 1060
called by restoreHistory, 839
gatherGlossLines, 1485
defun, 1485
gbc-time
called by statisticsInitialization, 1181
genCategoryTable
called by write-categorydb, 1176
genDomainTraceName, 59
called by domainToGenvar, 1054
called by genvar, 59
called by lassoc, 59
defun, 59
called by charDigitVal, 859
called by error, 859
called by gensymp, 859
called by pname, 859
defun, 859
called by dewritify, dewritifyInner, 854
called by breaklet, 86
called by gensymInt, 859
called by letPrint2, 1064
called by letPrint3, 1066
called by letPrint, 1062
called by spadTrace, isTraceable, 66
called by tracelet, 85
called by genDomainTraceName, 59
called by letPrint, 1062
called by augmentTraceNames, 65
called by clearCmdParts, 773
called by getAliasIfTracedMapParameter, 74
called by getBpiNameIfTracedMap, 75
called by getMapSig, 54
called by getMapSubNames, 65
called by getPreviousMapSubNames, 1056
called by importFromFrame, 19
called by isDomainValuedVariable, 1105
called by isInterpOnlyMap, 1058
called by isUncompiledMap, 1058
called by putHist, 832
called by restoreHistory, 839
calledby retract2Specialization, 704
calledby transTraceItem, 60
calledby writeHistModesAndValues, 845
get-current-directory, 260
calledby restart, 239
defun, 260
getAliasIfTracedMapParameter, 74
calledby mapLetPrint, 74
calls exit, 74
calls get, 74
calls isSharpVarWithNum, 74
calls seq, 75
calls substring, 75
uses $InteractiveFrame, 75
defun, 74
getalist
calledby displayProperties, sayFunctionDeps, 730
calledby importFromFrame, 19
calledby interpFunctionDepAlists, 737
calledby isExposedConstructor, 1026
calledby setExposeAddGroup, 89
calledby setExposeDropGroup, 95
getAndEvalConstructorArgument, 1104
calledby evaluateType1, 1080
calls compFailure, 1104
calls getValue, 1104
calls isLocalVar, 1104
calls isWrapped, 1104
calls mkObjWrap, 1104
calls objMode, 1104
calls objVal, 1104
calls timedEVALFUN, 1104
defun, 1104
getAndSay, 732
calledby displayProperties, 733
calls getL, 732
calls sayMSG, 732
defun, 732
getArgValue
calledby interpret1, 279
getBpiNameIfTracedMap, 75
calledby mapLetPrint, 74
calls exit, 75
calls get, 75
calls seq, 75
uses /tracenames, 75
uses $InteractiveFrame, 75
defun, 75
getBrowseDatabase, 1241
calls grepConstruct, 1241
calls member, 1241
uses $includeUnexposed?, 1241
defun, 1241
getCDTEntry
calledby dbShowCons1, 1547
getConstantFromDomain, 699
calledby coerceIntAlgebraicConstant, 698
calledby getConstantFromDomain, 699
calls compiledLookupCheck, 699
calls evalDomain, 699
calls getConstantFromDomain, 699
calls getOperationAlistFromLispLib, 699
calls isPartialMode, 699
calls lassoc, 699
calls opOf, 699
calls spadcall, 699
calls throwKeyedMsg, 699
defun, 699
getConstructorDocumentation, 1551
calledby dbShowConsDoc1, 1550
calls getdatabase, 1551
calls lassoc, 1551
calls qcaar, 1551
calls qcadar, 1551
calls qcar, 1551
calls qcdr, 1551
defun, 1551
getConstructorExports
calledby kePage, 1510
getConstructorForm
calledby augmentHasArgs, 1523
calledby conPageChoose, 1546
calledby dbAddDocTable, 1541
calledby dbShowConsDoc, 1549
calledby dbShowConstructorLines, 1554
calledby dbSpecialDescription, 1555
calledby dbSpecialExports, 1556
calledby dbSpecialOperations, 1556
calledby kcdePage, 1524
called by getConstructorDocumentation, 1080
getConstructor, 1156
getConstructorModemap
called by kArgPage, 1506
called by getConstructorSignature
called by dbShowConsDoc1, 1550
called by evaluateType1, 1080
called by reportOpsFromLispLib, 1024
called by valueArgsEqual?, 695
called by getConstructorModemap
called by kcuPage, 1525
called by make-databases, 1165
called by getConstructorModemap
called by kArgPage, 1506
called by getConstructorSignature
called by dbShowConsDoc1, 1550
called by evaluateType1, 1080
called by reportOpsFromLispLib, 1024
called by valueArgsEqual?, 695
called by getConstructorModemap
called by kcuPage, 1525
called by make-databases, 1165
called by getConstructorModemap
called by kArgPage, 1506
called by getConstructorSignature
called by dbShowConsDoc1, 1550
called by evaluateType1, 1080
called by reportOpsFromLispLib, 1024
called by valueArgsEqual?, 695
called by getConstructorModemap
called by kcuPage, 1525
called by make-databases, 1165
called by getConstructorModemap
called by kArgPage, 1506
called by getConstructorSignature
called by dbShowConsDoc1, 1550
called by evaluateType1, 1080
called by reportOpsFromLispLib, 1024
called by valueArgsEqual?, 695
called by getConstructorModemap
called by kcuPage, 1525
called by make-databases, 1165
called by getConstructorModemap
called by kArgPage, 1506
called by getConstructorSignature
called by dbShowConsDoc1, 1550
called by evaluateType1, 1080
called by reportOpsFromLispLib, 1024
called by valueArgsEqual?, 695
called by getConstructorModemap
called by kcuPage, 1525
called by make-databases, 1165
called by getConstructorModemap
called by kArgPage, 1506
called by getConstructorSignature
called by dbShowConsDoc1, 1550
called by evaluateType1, 1080
called by reportOpsFromLispLib, 1024
called by valueArgsEqual?, 695
called by getConstructorModemap
called by kcuPage, 1525
called by make-databases, 1165
called by getConstructorModemap
called by kArgPage, 1506
called by getConstructorSignature
called by dbShowConsDoc1, 1550
called by evaluateType1, 1080
called by reportOpsFromLispLib, 1024
called by valueArgsEqual?, 695
getdatabase, 1156
called by addoperations, 1153
called by coerceSubDomain, 700
called by dbAddDocTable, 1541
called by dbMkEvalable, 1529
called by dbSearchOrder, 1514
called by dbShowCons1, 1547
called by dbShowConsDoc1, 1550
called by describeSpad2Cmd, 331
called by displayOperationsFromLispLib, 1027
called by displayValue, 731
called by domainToGenvar, 1054
called by evaluateType1, 1080
called by getConstructorDocumentation, 1551
called by hasAtt, 654
called by initial-getdatabase, 1147
called by kCheckArgumentNumbers, 1531
called by kDomainName, 1528
called by kdPageInfo, 1505
called by libConstructorSig, 1563
called by loadLibNoUpdate, 1183
called by loadLib, 1181
called by local NRLib, 1161
called by mkEvalable, 1076
called by originsInOrder, 1540
called by reportOpsFromLispLib, 1024
called by reportOpsFromUnitDirectly, 1028
called by setExposeAddConstr, 92
called by setExposeDropConstr, 96
called by showdatabase, 1154
called by valueArgsEqual?, 695
called by whatConstructors, 1097
calls warn, 1156
uses *browse-stream*, 1156
uses *category-stream*, 1156
uses *default-domain-list*, 1156
uses *hasCategory-hash*, 1156
uses *hascategory-hash*, 1156
uses *interp-stream*, 1156
uses *miss*, 1156
uses *operation-hash*, 1156
uses *operation-stream*, 1156
uses $spadroot, 1156
defun, 1156
getDependentsOfConstructor, 1524
called by kcdePage, 1524
calls pathname, 1525
calls readLibPathFast, 1524
calls rread, 1525
calls rshut, 1525
defun, 1154
defun, 1130
getDirectoryList, 1130
called by getenviron, 255
defun, 255
called by getenviron, 255
called byDaaseName, 1169
called by buildHtMacroTable, 1333
called by copyright, 787
called by initial-getdatabase, 1147
called by initroot, 259
called by license, 866
called by make-databases, 1165
called by summary, 1035
called by getenviron, 255
defun, 255
called by geterFromDbL
called by getMsgInfoFromKey, 588
called by doSystemCommand, 718
called by stringSpaces, 741
called by subseq, 741
defun, 741
called by interpFunctionDepAlists, 737
defun, 1156
getenviron, 255
getenv
getenv, 255
geterFromDbL
called by getMsgInfoFromKey, 588
called by doSystemCommand, 718
called by stringSpaces, 741
called by subseq, 741
defun, 741
called by interpFunctionDepAlists, 737
defun, 1156
getFunctionFromDomain
called by coerceInt1, 675
getHtMacroItem, 1333
called by buildHtMacroTable, 1333
calls util.ht[7.1], 1334
defun, 1333
geti

called by displayProperties, 733
called by fetchOutput, 842
called by getAndSay, 732
getImports
called by kcnPage, 1526
getInfovec
called by hasAtt, 654
getInterpMacroNames
called by clearCmdParts, 773
called by displayMacros, 799
called by displayProperties, 733
called by displayWorkspaceNames, 726
getIProplist
called by displayProperties, 733
geti, 1200
called by coerceIntCommute, 689
called by coerceRetract, 709
called by dbShowConsDoc1, 1550
called by dbSpecialExports, 1556
called by dbSpecialOperations, 1556
called by reportOpsFromUnitDirectly, 1028
defun, 1200
getLinePos, 594
called by makeMsgFromLine, 593
defun, 594
getAddress, 594
called by makeMsgFromLine, 593
defun, 594
getMapSig, 54
called by saveMapSig, 53
calls boot-equal, 54
calls get, 54
uses $InteractiveFrame, 54
defun, 54
getMapSubNames, 65
called by traceSpad2Cmd, 47
calls getPreviousMapSubNames, 65
calls get, 65
calls unionq, 65
calls union, 65
uses /tracenames, 65
uses $InteractiveFrame, 65
uses $lastUntraced, 65
defun, 65
getMinimalVarMode
called by coerceInt, 673
getMsgArgL, 571
called by getMsgInfoFromKey, 588
called by sameMsg?, 598
defmacro, 571
getMsgCatAttr, 580
called by getMsgToWhere, 585
called by initToWhere, 590
called by msgImPr?, 580
called by msgNoRep?, 597
calls ifcdr, 580
calls ncAlist, 581
calls qassq, 581
defun, 580
getMsgFTTag
used by posPointers, 600
getMsgFTTag?, 581
called by getMsgPos2, 600
called by getMsgPos, 581
called by processChPosesForOneLine, 598
called by putFTText, 601
calls getMsgPosTagOb, 581
calls ifcar, 581
defun, 581
getMsgInfoFromKey, 588
called by putDatabaseStuff, 587
calls getErFromDbL, 588
calls getMsgArgL, 588
calls getMsgKey?, 588
calls getMsgKey, 588
calls removeAttributes, 588
calls segmentKeyedMsg, 588
calls substituteSegmentedMsg, 588
uses $msgDatabaseName, 588
defun, 588
getMsgKey, 570
called by getMsgInfoFromKey, 588
called by processKeyedError, 575
called by sameMsg?, 598
defmacro, 570
getMsgKey?, 584
called by getMsgInfoFromKey, 588

called by getStFromMsg, 577

calls identp, 584
defun, 584

getMsgPos, 581
called by erMsgCompare, 592
called by erMsgSep, 592
called by getPosStL, 578
called by posPointers, 600
called by processChPosesForOneLine, 598
called by processMsgList, 591
called by putFTText, 601
calls getMsgFTTag?, 581
calls getMsgPosTagOb, 581
defun, 581

getMsgPos2, 600
called by posPointers, 600
called by putFTText, 602
calls getMsgFTTag?, 600
calls getMsgPosTagOb, 601
calls ncBug, 601
defun, 600

getMsgPosTagOb, 570
called by getMsgFTTag?, 581
called by getMsgPos2, 601
called by getMsgPos, 581
defmacro, 570

getMsgPrefix, 571
called by processChPosesForOneLine, 598
defmacro, 571

getMsgPrefix?, 572
called by getStFromMsg, 576
called by processKeyedError, 575
called by tabbing, 584
defmacro, 572

getMsgTag, 572
called by getMsgTag?, 573
called by getStFromMsg, 576
called by initImPr, 589
called by leader?, 573
called by line?, 573
calls ncTag, 572
defmacro, 572

getMsgTag?, 572
called by processKeyedError, 575
calls getMsgTag, 573
calls ifcar, 572
defmacro, 572

getMsgText, 571
called by getStFromMsg, 576
called by putFTText, 601
defmacro, 571

getMsgToWhere, 585
called by toFile?, 585
called by toScreen?, 573
calls getMsgCatAttr, 585
defun, 585

getOperationAlistFromLisplib
called by getConstantFromDomain, 699
called by hasSig, 653
called by spadTrace, 67

getOpListForConstructorForm, 1030
called by reportOpsFromUnitDirectly, 1028
defun, 1030

getOpListWithUniqueSignatures, 1031
defun, 1031

ggetOption, 1068
called by spadTrace, 67
called by spadUntrace, 78
called by traceDomainConstructor, 71
calls assoc, 1068
defun, 1068

getParserMacroNames, 725
called by clearCmdParts, 773
called by displayMacros, 799
called by displayProperties, 733
called by displayWorkspaceNames, 726
uses $pfMacros, 725
defun, 725

getPosStL, 578
called by getStFromMsg, 576
calls decideHowMuch, 578
calls getMsgPos, 578
calls listDecideHowMuch, 578
calls msgImPr?, 578
calls ppos, 578
calls remFile, 578
calls remLine, 578
calls showMsgPos?, 578
uses $lastPos, 578
defun, 578

getPreStL, 577
called by getStFromMsg, 576
   calls size, 577
   uses $preLength, 577
defun, 577
getPreviousMapSubNames, 1056
called by getMapSubNames, 65
called by untraceMapSubNames, 1059
calls exit, 1056
calls get, 1056
calls seq, 1056
uses $InteractiveFrame, 1057
defun, 1056
getProplist, 1110
called by addBinding, 1109
called by getProplist, 1110
calls getProplist, 1110
calls search, 1110
uses $CategoryFrame, 1110
defun, 1110
getRefV16, 1249
defun, 1249
getRefV32, 1250
defun, 1250
getRefV8, 1248
defun, 1248
getspoolname, 883
called by regress, 882
defun, 883
getStFromMsg, 576
called by msgOutputter, 575
calls getMsgKey?, 577
calls getMsgPrefix?, 576
calls getMsgTag, 576
calls getMsgText, 576
calls getPosStL, 576
calls getPreStL, 576
calls pname, 577
calls tabbing, 577
defun, 576
defun, 576
getSubDomainPredicate, 701
called by coerceImmediateSubDomain, 701
called by getTraceOptions, 52
calls hput, 701
calls interpret, 701
calls mkAtree, 701
calls msubst, 701
calls removeZeroOne, 701
calls selectLocalMms, 701
calls transferPropsToNode, 701
uses $Boolean, 701
uses $InteractiveFrame, 701
uses $env, 701
uses $superHash, 701
defun, 701
getSystemCommandLine, 1038
called by synonymSpad2Cmd, 1037
calls strpos, 1038
calls substring, 1038
uses $currentLine, 1038
defun, 1038
getTraceOption, 55
called by getTraceOptions, 52
called by trace1, 48
calls concat, 55
calls exit, 55
calls getTraceOption,hn, 55
calls identp, 55
calls isListOfIdentifiersOrStrings, 55
calls isListOfIdentifiers, 55
calls object2String, 55
calls qcar, 55
calls qcdr, 55
calls selectOptionLC, 55
calls seq, 55
calls stackTraceOptionError, 55
calls throwKeyedMsg, 55
calls transOnlyOption, 55
uses $traceOptionList, 55
defun, 55
called by getTraceOption,hn, 54
called by getTraceOption, 55
calls domainToGenvar, 54
calls exit, 54
calls isDomainOrPackage, 54
calls seq, 54
calls stackTraceOptionError, 54
defun, 54
calls throwKeyedMsg, 52
calls throwListOfKeyedMsgs, 52
uses $traceErrorStack, 52
defun, 52
getUsersOfConstructor, 1526
calledby kcuPage, 1525
calls pathname, 1526
calls readLibPathFast, 1526
calls read, 1526
calls rshut, 1526
defun, 1526
getValue
calledby coerceInt1, 675
calledby evaluateType, 1078
calledby getAndEvalConstructorArgument, 1104
calledby interpret1, 279
getWorkspaceNames, 727
calledby displayProperties, 733
calledby displayWorkspaceNames, 727
calls umsort, 727
uses $InteractiveFrame, 727
defun, 727
glesseqp
calledby dbShowConstructorLines, 1554
grepConstruct
calledby getBrowseDatabase, 1241
grepSearchQuery
calledby constructorSearch, 1501
handleNoParseCommands, 717
calledby doSystemCommand, 718
calls concat, 742
calls member, 742
calls nplboot, 742
calls npload, 742
calls nploadym, 742
calls nploadyn, 742
calls nploadsys, 742
calls sayKeyedMsg, 742
calls stripLisp, 742
calls stripSpaces, 742
defun, 717
handleParsedSystemCommands, 740
calledby doSystemCommand, 718
calls dumbTokenize, 740
calls systemCommand, 740
defun, 740
handleTokenizeSystemCommands, 719
calledby doSystemCommand, 718
calls dumbTokenize, 719
calls systemCommand, 719
defun, 719
hasAtt, 654
calledby hasAttSig, 656
calledby hasCaty, 650
calls constructSubst, 654
calls getInfoc, 654
calls getdatabase, 654
calls hasCatExpression, 654
calls subCopy, 654
calls unifyStruct, 654
local def $domPvar, 654
defun, 654
hasAttSig, 656
calledby hasAttSig, 656
calledby hasCaty, 650
calls hasAtt, 656
calls hasSig, 656
calls keyedSystemError, 656
defun, 656
hasCat

calledby hasCaty, 650
hasCate, 663
calledby hasCatExpression, 657
calledby hasCate1, 657
calledby hasCateSpecial, 665
calledby hasCate1, 662
calledby hasSigAnd, 655
calledby hasSigOr, 656
calledby hasSig, 653
calls containsVariables, 663
calls hasCate1, 663
calls hasCateSpecial, 663
calls hasCate, 663
calls isPatternVar, 663
calls subCopy, 663
local def $hope, 664
local ref $EmptyMode, 663
local ref $Subst, 664
defun, 663
hasCate1, 657
called by hasCate, 663
calls hasCate, 657
local def $domPvar, 657
defun, 657
HasCategory
called by basicLookup, 1186
hascategory-hash
used by lefts, 1562
hasCateSpecial, 665
called by hasCate, 663
calls augmentSub, 665
calls canCoerceFrom, 665
calls containsVars, 665
calls eqcar, 665
calls hasCateSpecialNew, 665
calls hasCate, 665
calls hasCaty, 665
calls isSubDomain, 665
local ref $Integer, 665
local ref $QuotientField, 665
defun, 665
hasCateSpecialNew, 666
called by hasCateSpecial, 665
calls augmentSub, 666
calls defaultTargetFE, 666
calls eqcar, 666
calls hasCaty, 666
calls isEqualOrSubDomain, 666
calls member, 666
calls underDomainOf, 666
local ref $ComplexInteger, 666
local ref $Integer, 666
local ref $RationalNumber, 666
defun, 666
hasCatExpression, 657
called by hasAtt, 654
called by hasCatExpression, 657
calls hasCatExpression, 657
calls hasCate, 657
calls keyedSystemError, 657
defun, 657
hasCaty, 650
called by hasCateSpecialNew, 666
called by hasCateSpecial, 665
called by hasCate, 663
called by ofCategory, 649
calls augmentSub, 650
calls constructSubst, 650
calls domArg2, 650
calls domArg, 650
calls hasAttSig, 650
calls hasAtt, 650
calls hasCaty1, 651
calls hasCat, 650
calls hasSig, 650
calls mkDomPvar, 650
calls opOf, 650
calls subCopy, 650
calls unifyStruct, 651
local ref $domPvar, 651
defun, 650
hasCaty1, 662
called by hasCaty1, 662
called by hasCaty, 662
calls hasCate, 662
calls hasCaty1, 662
calls keyedSystemError, 662
local def $domPvar, 662
defun, 662
hasCorrectTarget, 685
defun, 685
hashable, 1246
calls Boolean, 1246
calls bpiname, 1247
calls compiledLookup, 1246
calls knownEqualPred, 1246, 1247
defun, 1246
hashCode?, 1185
called by basicLookupCheckDefaults, 1188
called by basicLookup, 1186
defmacro, 1185
hashString
called by basicLookupCheckDefaults, 1188
called by basicLookup, 1186
hashable-class
called by writify,writifyInner, 849
hashType
called by basicLookupCheckDefaults, 1188
called by basicLookup, 1186
hasIndent
called by mkDomTypeForm, 1507
hasNewInfoAlist
   called by kcPage, 1516
hasOptArgs?, 539
   called by pfApplication2Sex, 536
defun, 539
hasOption, 723
   called by trace1, 48
calls pname, 723
calls stringPrefix?, 723
defun, 723
hasPair, 1067
   called by hasPair, 1067
called by letPrint2, 1065
called by letPrint3, 1066
called by letPrint, 1062
calls hasPair, 1067
calls qcar, 1067
calls qcdr, 1067
defun, 1067
hasSig, 653
   called by hasAttSig, 656
called by hasCaty, 650
calls assq, 653
calls constructSubst, 653
calls constructor?, 653
calls getOperationAlistFromLisplib, 653
calls hasCaty, 653
calls hasSigAnd, 653
calls hasSigOr, 653
calls keyedSystemError, 653
calls subCopy, 653
calls unifyStruct, 653
local def $domPvar, 653
defun, 653
hasSigAnd, 655
   called by hasSigOr, 656
called by hasSig, 653
calls hasCate, 655
calls keyedSystemError, 655
calls subCopy, 655
defun, 655
hasSigOr, 656
   called by hasSig, 653
calls hasCate, 656
calls hasSigAnd, 656
calls keyedSystemError, 656
defun, 656
help, 810, 812
   called by helpSpad2Cmd, 812
defun, 812
manpage, 810
helpSpad2Cmd, 813
   called by abbreviationsSpad2Cmd, 755
called by frameSpad2Cmd, 10
called by help, 812
called by savesystem, 891
called by showSpad2Cmd, 1019
called by systemCommand, 720
calls newHelpSpad2Cmd, 813
calls sayKeyedMsg, 813
defun, 813
hget, 1194
   called by ScanOrPairVec, ScanOrInner, 858
called by conLowerCaseConTran, 1494
called by dbAddDocTable, 1541
called by dbDocTable, 1539
called by dbGetDocTable, 1543
called by dwritify, dwritifyInner, 854
called by kcPage, 1516
called by keyword?, 344
called by keyword, 344
called by saveDependentsHashTable, 1168
called by saveUsersHashTable, 1169
called by string2Constructor, 1494
called by writify, writifyInner, 849
defmacro, 1194
histFileErase, 860
   called by disableHist, 845
called by historySpad2Cmd, 822
called by initHist, 820
called by restoreHistory, 839
called by saveHistory, 837
called by setHistoryCore, 826
called by writeInputLines, 828
defun, 860
histFileName, 820
   called by addNewInterpreterFrame, 18
called by clearCmdAll, 771
called by disableHist, 845
called by historySpad2Cmd, 822
called by initHist, 820
called by readHiFi, 843
called by restoreHistory, 839
  called by saveHistory, 837
  called by setHistoryCore, 826
  called by writeHiFi, 844
  calls makeHistFileName, 820
  uses $interpreterFrameName, 820
defun, 820

histInputFileName, 820
called by saveHistory, 837
called by writeInputLines, 828
calls makePathname, 820
uses $historyDirectory, 820
uses $interpreterFrameName, 820
defun, 820

history, 815, 821
calls historySpad2Cmd, 821
calls sayKeyedMsg, 821
uses $options, 821
defun, 821
manpage, 815

historySpad2Cmd, 822
called by history, 821
calls changeHistListLen, 822
calls clearSpad2Cmd, 822
calls disableHist, 822
calls exit, 822
calls histFileErase, 822
calls histFileName, 822
calls initHistList, 822
calls member, 822
calls queryUserKeyedMsg, 822
calls resetInCoreHist, 822
calls restoreHistory, 822
calls saveHistory, 822
calls sayKeyedMsg, 822
calls selectOptionLC, 822
calls seq, 822
calls setHistoryCore, 822
calls showHistory, 822
calls string2id-n, 822
calls upcase, 822
calls writeInputLines, 822
uses $HiFiAccess, 822
uses $IOindex, 822
uses $options, 822
defun, 822

htkeys, 1195
called by lefts, 1562
called by saveDependentsHashTable, 1168
called by saveUsersHashTable, 1168
called by scanDictCons, 360
called by scanPunCons, 363
called by writify, writifyInner, 849
defun, 1195

hput, 1194
called by ScanOrPairVec, ScanOrInner, 858
called by addBinding, 1109
called by buildHtMacroTable, 1333
called by dbAddDocTable, 1541
called by dewritify, dewritifyInner, 854
called by getSubDomainPredicate, 701
called by writify, writifyInner, 849
defun, 1194

htAddHeading, 1415
called by kPage, 1497
defun, 1415

htAllOrNum, 1474
defun, 1474

htBcLinks, 1423
defun, 1423

htBcLispLinks, 1424
defun, 1424

htBcRadioButtons, 1427
defun, 1427

htBeginMenu
called by kcPage, 1516

htBeginTable
called by bcUnixTable, 1554

htBigSkip
called by kPage, 1510

htCacheAddChoice, 1471
defun, 1471

htCacheOne, 1474
defun, 1474

htCacheSet, 1473
defun, 1473

htCheck, 1463
defun, 1463

htCheckList, 1464
defun, 1464

htCopyProplist
called by dbShowCons1, 1547
INDEX

called by dbShowCons, 1545
called by kcPage, 1515
called by kccPage, 1522
called by kcpPage, 1519
called by kePage, 1510
called by kiPage, 1509
called by ksPage, 1513
htDoneButton, 1442
defun, 1442
htDoNothing, 1463
defun, 1463
htEndMenu
called by kcPage, 1516
htEndTable
called by bcUnixTable, 1555
htEscapeString, 1450
defun, 1450
htFunctionSetLiteral, 1460
defun, 1460
htGloss, 1482
defun, 1482
htGlossPage, 1482
defun, 1482
htGlossSearch, 1486
defun, 1486
htGreekSearch, 1487
defun, 1487
htInitPage, 1415
called by bcComplexLimit, 1377
called by bcDefiniteIntegrate, 1344
called by bcDifferentiate, 1349
called by bcDraw2DSolve, 1355
called by bcDraw2Dfun, 1352
called by bcDraw2Dpar, 1353
called by bcDraw3Dfun, 1357
called by bcDraw3Dpar1, 1362
called by bcDraw3Dpar, 1360
called by bcDraw, 1351
called by bcGen, 1398, 1399
called by bcIndefiniteIntegrate, 1343
called by bcInputEquations, 1385
called by bcInputExplicitMatrix, 1339
called by bcInputMatrixByFormula, 1337
called by bcInputSolveInfo, 1384
called by bcLaurentSeries, 1369
called by bcLimit, 1374
called by bcLinearSolveEqns, 1382
called by bcLinearSolveMatrix1, 1391
called by bcLinearSolveMatrixInhomo, 1392
called by bcLinearSolve, 1381
called by bcNotReady, 1401
called by bcProduct, 1348
called by bcPuiseuxSeries, 1371
called by bcReadMatrix, 1336
called by bcRealLimitGen, 1376
called by bcRealLimit, 1375
called by bcSeriesByFormula, 1366
called by bcSeriesExpansion, 1365
called by bcSeries, 1364
called by bcSolveEquationsNumerically, 1389
called by bcSolve, 1380
called by bcSum, 1346
called by bcSystemSolve, 1382
called by bcTaylorSeries, 1367
called by conOpPage1, 1533
called by constructorSearch, 1501
called by dbSpecialDescription, 1555
called by dbSpecialExports, 1556
called by dbSpecialOperations, 1556
called by kcPage, 1515
called by kccPage, 1522
called by kcpPage, 1519
called by kePage, 1510
called by kiPage, 1509
defun, 1415
htInitPageNoScroll
called by dbShowCons1, 1547
called by dbShowCons, 1545
called by kPage, 1497
called by ksPage, 1513
htInputStrings, 1429
defun, 1429
htKill, 1461
defun, 1461
htLispLinks, 1422
defun, 1422
htLispMemoLinks, 1423
defun, 1423
htMakeButton, 1438
defun, 1438
htMakeDoitButton, 1441
defun, 1441
htMakeDoneButton, 1437
called by bcDifferentiate, 1349
called by bcDraw2DSolve, 1355
called by bcDraw3Dpar1, 1362
called by bcInputEquations, 1386
called by bcInputExplicitMatrix, 1339
called by bcInputMatrixByFormula, 1337
called by bcLinearSolveEqns, 1382
called by bcSeriesExpansion, 1365
called by bcSolveEquationsNumerically, 1389
called by bcSystemSolve, 1383
defun, 1437

htMakeErrorPage, 1418
defun, 1418

htMakeInputList, 1448
defun, 1448

defun, 1472

htMakePage, 1416
called by bcComplexLimit, 1377
called by bcDefiniteIntegrate, 1344
called by bcDifferentiate, 1349
called by bcDraw2DSolve, 1355
called by bcDraw2Dfun, 1352
called by bcDraw2Dpar, 1354
called by bcDraw3Dfun, 1357
called by bcDraw3Dpar1, 1362
called by bcDraw3Dpar, 1360
called by bcGen, 1398, 1399
called by bcIndefiniteIntegrate, 1343
called by bcInputEquations, 1385
called by bcInputExplicitMatrix, 1339
called by bcInputMatrixByFormula, 1337
called by bcInputSolveInfo, 1384
called by bcLaurentSeries, 1369
called by bcLimit, 1374
called by bcLinearSolveEqns, 1382
called by bcLinearSolveMatrix1, 1391
called by bcLinearSolveMatrixInhomo, 1392
called by bcLinearSolve, 1381
called by bcNotReady, 1401
called by bcProduct, 1348
called by bcPuiseuxSeries, 1371
called by bcReadMatrix, 1336
called by bcRealLimitGen, 1376
called by bcRealLimit, 1375
called by bcSeriesByFormula, 1366
called by bcSeriesExpansion, 1365
called by bcSeries, 1364
called by bcSolveEquationsNumerically, 1389
called by bcSolve, 1380
called by bcSum, 1346
called by bcSystemSolve, 1382
called by bcTaylorSeries, 1367
called by bcUnixTable, 1554
called by kcPage, 1516
called by kdPageInfo, 1505
called by kePageDisplay, 1512
called by kePage, 1510
defun, 1416

defun, 1417

defun, 1406

defun, 1406

defun, 1435

defun, 1468

defun, 1468

defun, 1468

defun, 1468

defun, 1468

defun, 1402

defun, 1402

htpAddInputAreaProp, 1408
defun, 1408

defun, 1413

defun, 1406

htpDomainConditions, 1405
defun, 1405

defun, 1406

defun, 1405

htpDomainPvarSubstList, 1406

defun, 1406
INDEX

defun, 1405
htpInputAreaList, 1407
called by bcGenExplicitMatrix, 1341
called by bcLinearExtractMatrix, 1392
called by bcLinearMatrixGen, 1394
called by bcLinearSolveEqnsGen, 1396
defun, 1407
htpInputAreaList
called by bcInputSolveInfo, 1384
defun, 1411
htpLabelDefault, 1411
defun, 1411
htpLabelErrorMsg, 1411
defun, 1411
htpLabelFilter, 1412
defun, 1412
htpLabelFilteredInputString, 1409
defun, 1409
htpLabelInputString, 1409
called by bcInputExplicitMatrix, 1339
called by bcInputMatrixByFormula, 1337
defun, 1410
htpLabelType, 1411
defun, 1411
htpName, 1404
defun, 1404
htpPageDescription, 1412
defun, 1412
htpProperty, 1408
called by bcGenExplicitMatrix, 1341
called by bcInputMatrixByFormulaGen, 1338
called by bcLinearSolveMatrixInhomo, 1392
called by bcMatrixGen, 1341
called by bcRealLimitGen1, 1377
called by bcSolveEquations, 1390
called by conOpPage, 1532
called by dbCompositeWithMap, 1534
called by dbConsHeading, 1553
called by dbSelectCon, 1552
called by dbShowConditions, 1552
called by dbShowCons1, 1547
called by dbShowConsDoc1, 1550
called by dbShowConsDoc, 1549
called by dbShowConsKindsFilter, 1549
called by dbShowCons, 1545
called by kArgPage, 1506
called by kcPage, 1515
called by kcaPage1, 1521
called by kccPage, 1522
called by kcdePage, 1524
called by kcnPage, 1526
called by kcpPage, 1518
called by kcuPage, 1525
called by kePage, 1509
called by kiPage, 1508
called by koPageAux1, 1537
called by koPageFromKKPage, 1536
called by koPageInputAreaUnchanged?, 1527
called by koaPageFilterByName, 1537
defun, 1409
htpLabelSpadType, 1412
defun, 1412
htpLabelSpadValue, 1410
called by bcInputExplicitMatrix, 1339
called by bcInputMatrixByFormula, 1337
defun, 1410
htpLabelType, 1411
defun, 1411
htpName, 1404
defun, 1404
htpPageDescription, 1412
defun, 1412
htpPropertyList, 1408
called by bcInputEquations, 1385
called by bcInputExplicitMatrix, 1339
called by bcInputSolveInfo, 1384
called by bcLinearSolveMatrixInhomo, 1392
called by bcSolveEquationsNumerically, 1389
defun, 1408
htpRadioButtonAlist, 1406
defun, 1406
htProcessBcButtons, 1420
defun, 1420
htProcessBcStrings, 1421
defun, 1421
htProcessDoitButton, 1441
defun, 1441
htProcessDomainConditions, 1431
defun, 1431
htProcessDoneButton, 1438
defun, 1438
htProcessToggleButtons, 1419
defun, 1419
htProperty
called by bcInputEquations, 1385
defun, 1405
htpSetDomainConditions, 1405
defun, 1405
htpSetDomainPvarSubstList, 1406
defun, 1406
htpSetDomainVariableAlist, 1405
defun, 1405
htpSetInputAreaAlist, 1407
defun, 1407
htpSetLabelErrorMsg, 1411
defun, 1411
htpSetLabelInputString, 1410
defun, 1410
htpSetLabelSpadValue, 1410
defun, 1410
htpSetName, 1404
defun, 1404
htpSetPageDescription, 1413
defun, 1413
htpSetProperty, 1408
called by bcGenExplicitMatrix, 1341
called by bcInputEquations, 1385
called by bcInputExplicitMatrix, 1339
called by bcInputMatrixByFormula, 1337
called by bcInputSolveInfo, 1384
called by bcLinearSolveEqns, 1384
called by bcLinearSolveMatrixInhomo, 1392
called by bcReadMatrix, 1336
called by bcRealLimitGen, 1376
called by bcSolveSingle, 1383
called by bcSystemSolveEqns, 1384
called by conOpPage1, 1533
called by constructorSearch, 1501
called by dbShowCons1, 1547
called by dbShowConsKindsFilter, 1549
called by dbShowCons, 1545
called by dbSpecialDescription, 1555
called by dbSpecialOperations, 1556
called by kArgPage, 1506
called by kDomainName, 1528
called by kPage, 1497
called by kcPage, 1515
called by kcaPage1, 1521
called by kccPage, 1522
called by kcdePage, 1524
called by kcnPage, 1526
called by kcpPage, 1519
called by kcuPage, 1525
called by kePageDisplay, 1512
called by kePage, 1509, 1510
called by koPageAux, 1536
called by koPage, 1535
called by koaPageFilterByName, 1538
called by ksPage, 1513
defun, 1408
htpSetRadioButtonAlist, 1407
defun, 1407
htQuery
called by constructorSearch, 1501
htQuote, 1418
defun, 1418
htRadioButtons, 1426
defun, 1426
htSay
called by bcInputEquations, 1385
called by bcUnixTable, 1554
called by dbConsExposureMessage, 1549
called by kPage, 1497
called by kcPage, 1515
called by kdPageInfo, 1505
called by kePageDisplay, 1512
called by kePage, 1510
called by ksPage, 1513
htSayHrule
called by `dbShowConditions`, 1552
`htSayStandard`
called by `dbShowCons1`, 1547
called by `kPage`, 1497
called by `kcPage`, 1516
called by `kdPageInfo`, 1505
called by `kePageDisplay`, 1512
called by `kePage`, 1510
called by `ksPage`, 1513
`htSetCache`, 1470
defun, 1470
`htSetExpose`, 1470
defun, 1470
`htSetFunCommand`, 1461
defun, 1461
`htSetFunCommandContinue`, 1461
defun, 1461
`htSetHistory`, 1469
defun, 1469
`htSetInputLibrary`, 1469
defun, 1469
`htSetInteger`, 1458
defun, 1458
`htSetLinkerArgs`, 1470
defun, 1470
`htSetLiteral`, 1456
defun, 1456
`htSetLiterals`, 1455
defun, 1455
`htSetNotAvailable`, 1462
defun, 1462
`htSetOutputCharacters`, 1470
defun, 1470
`htSetOutputLibrary`, 1469
defun, 1469
`htSetSystemVariable`, 1482
defun, 1482
`htSetSystemVariableKind`, 1481
defun, 1481
`htSetvarDoneButton`, 1460
defun, 1460
`htSetVars`, 1450
defun, 1450
`htShowCount`, 1453
defun, 1453
`htShowFunctionPage`, 1458
defun, 1458
`htShowFunctionPageContinued`, 1459
defun, 1459
`htShowIntegerPage`, 1457
defun, 1457
`htShowLiteralsPage`, 1455
defun, 1455
`htShowPage`, 1416
called by `bcComplexLimit`, 1377
called by `bcDefiniteIntegrate`, 1344
called by `bcDifferentiate`, 1349
called by `bcDraw2DSolve`, 1356
called by `bcDraw2Dfun`, 1352
called by `bcDraw2Dpar`, 1354
called by `bcDraw3DFun`, 1358
called by `bcDraw3Dpar1`, 1362
called by `bcDraw3Dpar`, 1360
called by `bcDraw`, 1351
called by `bcGen`, 1398, 1399
called by `bcIndefiniteIntegrate`, 1343
called by `bcInputEquations`, 1386
called by `bcInputExplicitMatrix`, 1339
called by `bcInputMatrixByFormula`, 1337
called by `bcInputSolveInfo`, 1384
called by `bcLaurentSeries`, 1369
called by `bcLimit`, 1374
called by `bcLinearSolveEquns`, 1382
called by `bcLinearSolveMatrix1`, 1391
called by `bcLinearSolveMatrixInhomo`, 1392
called by `bcLinearSolve`, 1381
called by `bcNotReady`, 1401
called by `bcProduct`, 1348
called by `bcPuiseuxSeries`, 1371
called by `bcReadPuiseuxSeries`, 1336
called by `bcRealLimitGen`, 1376
called by `bcRealLimit`, 1375
called by `bcSeriesByFormula`, 1366
called by `bcSeriesExpansion`, 1365
called by `bcSeries`, 1364
called by `bcSolveEquationsNumerically`, 1389
called by `bcSolve`, 1380
called by `bcSum`, 1346
called by `bcSystemSolve`, 1383
called by `bcTaylorSeries`, 1367
called by `constructorSearch`, 1501
called by `dbSpecialDescription`, 1555
called by dbSpecialExports, 1556
called by kcPage, 1516
called by kiPage, 1509
defun, 1416
htShowPageNoScroll, 1416
called by dbShowCons1, 1547
called by kPage, 1497
defun, 1416
htShowSetPage, 1454
defun, 1454
htShowSetTree, 1451
defun, 1451
htShowSetTreeValue, 1453
defun, 1453
htSowPage
called by kePage, 1510
htStringPad, 1402
calls strconc, 1402
defun, 1402
htsv, 1450
defun, 1450
htSystemVariables, 1479
defun, 1479
htSystemVariables,displayOptions, 1477
defun, 1477
htSystemVariables,fn, 1477
defun, 1477
htSystemVariables,functionTail, 1478
defun, 1478
htSystemVariables,gn, 1476
defun, 1476
htTextSearch, 1489
defun, 1489
htTutorialSearch, 1491
defun, 1491
idChar?, 351
called by scanW, 351
defmacro, 351
identp, 1197
called by countCache, 912
called by downcase, 1206
called by getMsgKey?, 584
called by getTraceOption, 55
called by isDomainValueVariable, 1105
called by isListOfIdentiifiersOrStrings, 1056
called by isListOfIdentiifiers, 1055
called by isSharpVar, 1064
called by isgenvar, 1064
called by messageprint-1, 1199
called by nCAlist, 637
called by ncTag, 637
called by ofCategory, 649
called by restoreHistory, 839
called by rwrite, 847, 848
called by selectOption, 751
called by undo, 36
called by upcase, 1206
defmacro, 1197
if, 617
syntax, 617
If?, 304
called by include1, 307
calls quotient, 304
defun, 304
ifcar
called by conLowerCaseConTran, 1494
called by conOpPage1, 1533
called by conSpecialString?, 1502
called by dbShowCons, 1545
called by defaultTargetFE, 668
called by fillerSpaces, 242
called by getMsgFTTag?, 581
called by getMsgTag?, 572
called by pfAbSynOp, 634
called by pfExpression, 494
called by pfLeaf, 477
called by pfSymbol, 481
called by pfSynb, 481
called by posPointers, 600
called by remLine, 579
called by string2Constructor, 1494
called by tokConstruct, 633
ifcdr
called by clearParserMacro, 725
called by conOpPage1, 1533
called by domainDescendantsOf, 1507
called by getMsgCatAttr, 580
called by kePage, 1510
called by mac0Get, 456
called by remFile, 579
called by reportOpsFromLisplib, 1024
INDEX

called by setMsgCatlessAttr, 587
called by specialChar, 1126

ifCond, 314
called by incLude1, 307
calls MakeSymbol, 314
calls incCommandTail, 314
uses $inclAssertions, 314
defun, 314

IfKeepPart, 302
defvar, 302

IfSkipPart, 302
defvar, 302

IfSkipToEnd, 302
defvar, 302

ignorep, 888
called by split, 886
defun, 888

iht, 1413
defun, 1413

images

Restart, 238

import
called by frameSpad2Cmd, 11

importFromFrame, 19
called by frameSpad2Cmd, 11
called by importFromFrame, 19
calls boot-equal, 19
calls clearCmdParts, 19
calls exit, 19
calls frameEnvironment, 19
calls frameNames, 19
calls framename, 19
calls getalist, 19
calls get, 19
calls importFromFrame, 19
calls member, 19
calls putHist, 19
calls queryUserKeyedMsg, 19
calls sayKeyedMsg, 19
calls seq, 19
calls string2id-n, 19
calls throwKeyedMsg, 19
calls upcase, 19
uses $interpreterFrameRing, 19
defun, 19

in-stream, 1107

used by ncTopLevel, 248
used by serverReadLine, 269
defun, 1107

incActive?, 328
called by incLude1, 306
defun, 328

incAppend, 312
called by incAppend1, 312
called by incLude1, 306
called by next1, 263
calls Delay, 312
calls incAppend1, 312
defun, 312

incAppend1, 312
called by incAppend, 312
calls StreamNull, 312
calls incAppend, 312
defun, 312

incBiteOff, 862
called by incFileName, 862
defun, 862

incClassify, 324
called by incLude1, 307
calls incCommand?, 325
uses incCommands, 325
defun, 324

incCommand?, 326
called by incClassify, 325
defun, 326

incCommands, 324
used by incClassify, 325
defun, 324

incCommandTail, 327
called by assertCond, 321
called by ifCond, 314
called by incLude1, 306
called by incFname, 327
calls incDrop, 327
defun, 327

incConsoleInput, 328
called by incLude1, 306
calls incRgen, 328
calls make-instream, 328
defun, 328

incDrop, 327
called by incCommandTail, 327
calls substring, 327
defun, 327
incFileInput, 327
called by incLude1, 306
calls incRgen, 327
calls make-instream, 327
defun, 327
incFileName, 862
called by inclFname, 327
called by ncloopIncFileName, 861
calls incBiteOff, 862
defun, 862
incHandleMessage, 314
called by incRenumberLine, 300
calls ncBug, 301
calls ncSoftError, 301
defun, 301
inclIgen, 299
called by inclIgen1, 300
called by incRenumber, 298
calls Delay, 299
calls inclIgen1, 299
defun, 299
inclIgen1, 300
called by inclIgen, 299
calls inclIgen, 300
defun, 300
inclFname, 327
called by inclLude1, 306
calls incCommandTail, 327
calls incFileName, 327
defun, 297
inclLine, 312
called by xlMsg, 311
called by xISkip, 314
calls inclLine1, 312
defun, 312
inclLine1, 313
called by inclLine, 312
called by xlOK1, 311
calls lnCreate, 313
defun, 313
inclmsgCannotRead, 317
called by xlCannotRead, 316
calls thefname, 317
defun, 317
inclmsgCmdBug, 323
called by xlCmdBug, 323
defun, 323
inclmsgConActive, 318
called by xlConActive, 318
calls theid, 318
defun, 318
inclmsgConsole, 320
called by xlConsole, 319
defun, 320
inclmsgConStill, 319
called by xlConStill, 319
calls theid, 319
defun, 319
inclmsgFileCycle, 317
called by xlFileCycle, 317
calls porigin, 317
calls theid, 317
defun, 317
inclmsgFinSkipped, 320
called by xlSkippingFin, 320
defun, 320
inclmsgIfBug, 323
called by xlIfBug, 323
defun, 323
inclmsgIfSyntax, 322
called by xlIfSyntax, 322
calls concat, 322
calls theid, 322
calls theorigin, 322
defun, 322
inclmsgNoSuchFile, 316
called by xlNoSuchFile, 315
calls thefname, 316
defun, 316
inclmsgPrematureEOF, 313
called by xlPrematureEOF, 310
calls theorigin, 313
defun, 313
inclmsgPrematureFin, 321
called by xlPrematureFin, 321
calls theorigin, 321
defun, 321
inclmsgSay, 315
called by xlSay, 315
calls theid, 315
defun, 315
incLude, 301
called by incLude1, 306
called by incStream, 298
called by incString, 263
calls Delay, 301
defun, 301
include, 861
calls incLude1, 301
manpage, 861
incLude1, 306
called by include, 301
calls Elseif?, 307
calls If?, 307
calls KeepPart?, 307
calls Rest, 306
calls SkipEnd?, 307
calls SkipPart?, 307
calls Skipping?, 306
calls StreamNull, 306
calls Top?, 306
calls assertCond, 307
calls concat, 306
calls expand-tabs, 307
calls ifCond, 307
calls incActive?, 306
calls incAppend, 306
calls incClassify, 307
calls incCommandTail, 306
calls incConsoleInput, 306
calls incFileInput, 306
calls incLude, 306
calls incNConsoles, 307
calls incFname, 306
calls xICannotRead, 306
calls xICmdBug, 307
calls xIConActive, 306
calls xIConStill, 306
calls xIConsole, 307
calls xIFileCycle, 306
calls xIIIfBug, 307
calls xIIIfSyntax, 307
calls xINoSuchFile, 306
calls xIOK1, 306
calls xIOK, 306
calls xIPrematureEOF, 306
calls xIPrematureFin, 307
calls xISay, 306
calls xISkippingFin, 307
calls xISkip, 306
defun, 306
incNConsoles, 328
called by incLude1, 307
called by incNConsoles, 328
calls incNConsoles, 328
defun, 328
incPrefix?, 326
called by lineoftoks, 335
called by scanIgnoreLine, 336
defun, 326
incRenumber, 298
called by incStream, 298
called by incString, 263
calls incIGen, 298
calls incZip, 298
defun, 298
incRenumberItem, 300
called by incRenumberLine, 300
calls lnSetGlobalNum, 300
defun, 300
incRenumberLine, 300
calls incHandleMessage, 300
calls incRenumberItem, 300
defun, 300
incRgen, 329
called by incConsoleInput, 328
called by incFileInput, 327
called by incRgen1, 330
called by incStream, 298
calls Delay, 329
calls incRgen1, 329
defun, 329
incRgen1, 330
called by incRgen, 329
calls incRgen, 330
uses StreamNil, 330
defun, 330
incStream, 298
called by intloopInclude0, 288
called by ncloopInclude0, 297
calls incLude, 298
calls incRenumber, 298
calls incRgen, 298
uses Top, 298
defun, 298
incString, 263
called by intloopProcessString, 261
called by parseFromString, 273
called by parseNoMacroFromString, 1531
calls incLude, 263
calls incRenumber, 263
uses Top, 263
defun, 263
incZip, 298
called by incRenumber, 298
called by incZip1, 299
calls Delay, 299
calls incZip1, 299
defun, 298
incZip1, 299
called by incZip, 299
calls StreamNull, 299
calls incZip, 299
defun, 299
infgeneric, 333
defvar, 333
init-boot/spad-reader, 1114
calls ioclear, 1114
calls next-lines-clear, 1114
uses *standard-output*, 1114
uses $spad-errors, 1114
uses boot-line-stack, 1114
uses file-closed, 1114
uses line-handler, 1114
uses meta-error-handler, 1114
uses spaderrorstream, 1114
uses xtokenreader, 1114
defun, 1114
init-memory-config, 258
called by restart, 239
calls allocate-contiguous-pages, 258
calls allocate-relocatable-pages, 258
calls allocate, 258
calls set-hole-size, 258
defun, 258
initHist, 820
called by restart, 239
calls $replace, 820
calls histFileErase, 820
calls histFileName, 820
calls initHistList, 820
calls makeInputFilename, 820
calls oldHistFileName, 820
uses $HiFiAccess, 820
uses $useInternalHistoryTable, 820
defun, 820
initHistList, 821
called by addNewInterpreterFrame, 18
called by historySpad2Cmd, 822
called by initHist, 820
uses $HistListAct, 821
uses $HistListLen, 821
uses $HistList, 821
uses $HistRecord, 821
defun, 821
initial-getdatabase, 1147
called by resethashstables, 1146
calls getdatabase, 1147
calls getenviron, 1147
defun, 1147
initializeInterpreterFrameRing, 12
called by restart, 239
calls emptyInterpreterFrame, 12
calls updateFromCurrentInterpreterFrame, 12
uses $interpreterFrameName, 12
uses $interpreterFrameRing, 12
defun, 12
initializeSetVariables, 894
called by initializeSetVariables, 894
called by resetWorkspaceVariables, 894
calls initializeSetVariables, 894
calls literals, 894
calls sayMSG, 894
calls translateYesNo2TrueFalse, 894
calls tree, 894
defun, 894
initializeTimedNames
called by processInteractive, 274
initImPr, 589
called by msgCreate, 570
calls getMsgTag, 589
calls setMsgUnforcedAttr, 589
uses $erMsgToss, 589
uses $imPrTagGuys, 589
defun, 589
initroot, 259
called by restart, 239
calls getenviron, 259
calls reroot, 259
uses $spadroot, 259
defun, 259
initToWhere, 590
called by msgCreate, 570
calls getMsgCatAttr, 590
calls setMsgUnforcedAttr, 590
defun, 590
input-libraries, 907
used by addInputLibrary, 907
used by dropInputLibrary, 908
used by openOutputLibrary, 905
used by setInputLibrary, 906
defvar, 907
insert
called by originsInOrder, 1540
called by updateSourceFiles, 806
insertAlist, 913
called by countCache, 912
called by kePageOpAlist, 1511
calls ?order, 913
calls rplac, 913
defun, 913
insertpile, 557
called by intloopInclude0, 288
called by nloopInclude0, 297
calls npNull, 557
calls pileCforest, 557
calls pilePlusComments, 557
calls pilePlusComment, 557
calls pileTree, 557
defun, 557
insertPos, 601
called by posPointers, 600
calls done, 601
defun, 601
installConstructor
called by loadLibNoUpdate, 1183
called by loadLib, 1181
called by localrnlib, 1161
intCodeGenCOERCE
called by coerceInt0, 674
integer-decode-float-denominator, 1220
defun, 1220
integer-decode-float-exponent, 1220
defun, 1220
integer-decode-float-quotient, 1219
defun, 1219
integer-decode-float-sign, 1220
defun, 1220
intern
called by dbShowConstructorLines, 1554
intern

called by constructSubst, 664
called by countCache, 912
called by spadTraceAlias, 76
InterpExecuteSpadSystemCommand, 256
called by ExecuteInterpSystemCommand, 256
uses $intCoerceFailure, 256
uses $intSpadReader, 256
catches, 256
defun, 256
interpreFunctionDepAlists, 737
called by displayProperties, 733
calls getFlag, 737
calls getalist, 737
calls putalist, 737
uses $InteractiveFrame, 737
uses $dependeeAlist, 737
uses $dependentAlist, 737
uses $e, 737
defun, 737
interpopen, 1149
called by resethashables, 1146
called by restart0, 241
calls DaaseName, 1150
calls make-database, 1150
uses *allconstructors*, 1150
uses *interp-stream*, 1150
uses *interp-stream-stamp*, 1150
uses $spadroot, 1150
defun, 1149
interpret, 278
called by getSubDomainPredicate, 701
called by interpretTopLevel, 278
calls interpret1, 278
uses $env, 278
<table>
<thead>
<tr>
<th>Function</th>
<th>Line</th>
<th>Calls To</th>
</tr>
</thead>
<tbody>
<tr>
<td>uses $eval, uses $genValue, defun</td>
<td>279</td>
<td></td>
</tr>
<tr>
<td>interpret1, called by interpret</td>
<td>278</td>
<td></td>
</tr>
<tr>
<td>calls bottomUp, calls getArgValue, calls getVal, calls interpret2, calls keyedSystemError, calls mkAtreeWithSrcPos, calls putTarget, uses $eval, uses $genValue, defun</td>
<td>279</td>
<td></td>
</tr>
<tr>
<td>interpret2, called by interpret1</td>
<td>279</td>
<td></td>
</tr>
<tr>
<td>calls coerceInteractive, calls member, calls mkObj, calls objMode, calls objVal, calls systemErrorHere, calls throwKeyedMsgCannotCoerceWith-Value, uses $EmptyMode, uses $ThrowAwayMode, defun</td>
<td>280</td>
<td></td>
</tr>
<tr>
<td>interpretTopLevel, called by interpretTopLevel, called by processInteractive1, calls interpret, calls peekTimedName, calls stopTimingProcess, uses $timedNameStack, catches, defun</td>
<td>278</td>
<td></td>
</tr>
<tr>
<td>interrupt, called by break,</td>
<td>87</td>
<td></td>
</tr>
<tr>
<td>intInterpretPform, called by phInterpret, calls pf2Sex, calls processInteractive, calls zeroOneTran, defun</td>
<td>292</td>
<td></td>
</tr>
<tr>
<td>intloop, called by ncIntLoop,</td>
<td>249</td>
<td></td>
</tr>
<tr>
<td>calls SpadInterpretStream, calls resetStackLimits, uses $IntRestart, uses $IntTopLevel, catches, defun</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>intloopEchoParse, called by intloopInclude0, calls dqToList, calls mkLineList, calls ncloopDQlines, calls ncloopPrintLines, calls npParse, calls setCurrentLine, uses $EchoLines, uses $lines, defun</td>
<td>294</td>
<td></td>
</tr>
<tr>
<td>intloopInclude, called by SpadInterpretStream, called by intloopInclude0, defun</td>
<td>288</td>
<td></td>
</tr>
<tr>
<td>intloopInclude0, called by intloopInclude0, called by intloopProcessString, called by intloopProcess, calls StreamNull, calls $systemCommandFunction, calls intloopProcess, calls intloopSpadProcess, calls pfAbSynOp?, defun</td>
<td>261</td>
<td></td>
</tr>
</tbody>
</table>
calls setCurrentLine, 288
  calls tokPart, 288
  uses $systemCommandFunction, 289
defun, 288
intloopProcessString, 261
calledby intloopReadConsole, 254
calls incString, 261
calls intloopProcess, 261
calls next, 261
calls setCurrentLine, 261
defun, 261
intloopReadConsole, 253
calledby SpadInterpretStream, 252
calledby intloopReadConsole, 253
calls concat, 253
calls intloopPrefix?, 253
calls intloopProcessString, 254
calls intloopReadConsole, 253
calls intnplisp, 253
calls leaveScratchpad, 253
calls mkprompt, 253
calls mkprompt, 253
calls ncPutQ, 289
calls $NeedToSignalSessionManager, 289
calls $currentCarrier, 289
calls $intCoerceFailure, 289
calls $intSpadReader, 289
calls $ncMsgList, 289
calls $prevCarrier, 289
calls $stepNo, 289
calls flung, 289
catches, 289
defun, 289
intloopSpadProcess,interp, 289
calledby intloopSpadProcess, 289
calls ncConversationPhase, 290
calls ncEltQ, 290
calls ncError, 290
defun, 290
intnplisp, 260
calledby intloopReadConsole, 253
calls nplisp, 260
calls $currentLine, 260
defun, 260
intp
calledby dewritify,dewritifyInner, 854
intProcessSynonyms, 257
calledby ExecuteInterpSystemCommand, 256
calls processSynonyms, 257
uses line, 257
defun, 257
ioClear, 1118
calledby init-boot/spad-reader, 1114
called by spad-syntax-error, 1115
calls line-clear, 1118
calls reduce-stack-clear, 1118
uses $boot, 1118
uses $spad, 1118
uses current-fragment, 1118
uses current-line, 1118
defun, 1118
ioStat, 1116
calledby spad-long-error, 1115
calls line-past-end-p, 1116
calls line-print, 1116
calls next-lines-show, 1116
calls token-stack-show, 1116
uses $boot, 1116
uses $spad, 1116
uses $current-line, 1116
defun, 1116
is-console[9]
called by serverReadLine, 269, 1159
called by shut, 1128
isDomain
called by ?t, 84
called by addTraceItem, 83
called by compiledLookup, 1185
isDomainOrPackage, 1061
called by /tracereply, 77
called by \(?t, \ 84
called by addTraceItem, \ 83
called by getTraceOption, \hn, \ 54
called by prTraceNames, \fn, \ 80
called by shortenForPrinting, \ 1067
called by spadReply, printName, \ 1069
called by spadTrace, \ 67
called by traceReply, \ 81
called by untraceDomainConstructor, keepTraced?, \ 73
called by writify, writifyInner, \ 849
calls isFunctor, \ 1061
calls opOf, \ 1061
calls poundsign, \ 1061
calls refvecp, \ 1061
defun, \ 1061
isDomainValuedVariable, \ 1105
called by evaluateType, \ 1078
called by reportOperations, \ 1021
calls get, \ 1105
calls identp, \ 1105
calls member, \ 1105
calls objMode, \ 1105
calls objValUnwrap, \ 1105
local ref $\texttt{InteractiveFrame}, \ 1105
local ref $\texttt{env}, \ 1105
local ref $\texttt{e}, \ 1105
defun, \ 1105
isEqualOrSubDomain, \ 668
called by coerceInt1, \ 675
called by coerceRetract, \ 709
called by defaultTargetFE, \ 667
called by hasCateSpecialNew, \ 666
calls isSubDomain, \ 668
defun, \ 668
isExposedConstructor, \ 1026
called by conOpPage1, \ 1533
called by dbShowCons1, \ 1547
called by dbShowConsDoc1, \ 1550
called by kPage, \ 1497
called by reportOpsFromLisplib, \ 1024
called by reportOpsFromUnitDirectly, \ 1028
calls getalist, \ 1026
local ref $\texttt{globalExposureGroupAlist}, \ 1026
local ref $\texttt{localExposureData}, \ 1026
defun, \ 1026
isFunctor
called by funfind, LAM, \ 1060
called by isDomainOrPackage, \ 1061
called by trace1, \ 48
called by traceReply, \ 81
isgenvar, \ 1064
called by \(?t, \ 84
called by letPrint2, \ 1064
called by letPrint3, \ 1066
called by letPrint, \ 1062
called by traceReply, \ 81
called by tracelet, \ 85
calls digitp, \ 1064
calls identp, \ 1064
calls size, \ 1064
defun, \ 1064
isIntegerString, \ 740
called by tokTran, \ 739
defun, \ 740
isInternalMapName
called by displayProperties, \ 733
isInterpMacro
called by displayMacro, \ 726
called by displayProperties, \ 733
isInterpOnlyMap, \ 1058
calls get, \ 1058
uses $\texttt{InteractiveFrame}, \ 1058
defun, \ 1058
isLegitimateMode
called by coerceCommuteTest, \ 690
called by coerceIntTableOrFunction, \ 690
isListOfIdentifiers, \ 1055
called by getTraceOption, \ 55
calls exit, \ 1055
calls identp, \ 1055
calls seq, \ 1055
defun, \ 1055
isListOfIdentifiersOrStrings, \ 1056
called by getTraceOption, \ 55
calls exit, \ 1056
calls identp, \ 1056
calls seq, \ 1056
defun, \ 1056
called by getAndEvalConstructorArgument, 1104
isMap
  called by clearCmdParts, 773
isNameOfType
  called by reportOperations, 1021
isNewWorldDomain
  called by basicLookup, 1186
isPartialMode, 650
  called by evalCategory, 1105
called by getConstantFromDomain, 699
called by retract2Specialization, 704
calls contained, 650
  local ref $EmptyMode, 650
defun, 650
isPatternVar, 661
  called by containsVars1, 661
called by containsVars, 661
called by hasCate, 663
called by unifyStructVar, 659
called by unifyStruct, 658
defun, 661
isRectangularList
  called by retract2Specialization, 704
isSharpVar, 1064
  called by domArg2, 652
called by isSharpVarWithNum, 1063
calls identp, 1064
defun, 1064
isSharpVarWithNum, 1063
  called by getAliasIfTracedMapParameter, 74
called by letPrint2, 1064
called by letPrint3, 1066
called by letPrint, 1062
calls dig2fix, 1063
calls digitp, 1063
calls isSharpVar, 1063
calls pname, 1063
calls qsize, 1063
defun, 1063
isSubDomain
  called by coerceInt1, 675
called by coerceIntByMap, 693
called by hasCateSpecial, 665
called by isEqualOrSubDomain, 668
calls domainDescendantsOf, 1506
calls getConstructorModemap, 1506
calls htpProperty, 1506
calls htpSetProperty, 1506
calls mkDomTypeForm, 1506
calls position, 1506
calls sublisFormal, 1506
defun, 1506
kArgumentCheck, 1529
  calledby kDomainName, 1528
calls conSpecialString?, 1529
calls form2String, 1529
calls opOf, 1529
defun, 1529
kcaPage, 1520
calls ancestorsOf, 1520
calls kcaPage1, 1520
defun, 1520
kcaPage1, 1521
  calledby kcaPage, 1520
calledby kcdPage, 1520
calls augmentHasArgs, 1521
calls dbShowCons, 1521
calls errorPage, 1521
calls form2String, 1521
calls function, 1521
calls htpProperty, 1521
calls htpSetProperty, 1521
calls kDomainName, 1521
calls listSort, 1521
calls opOf, 1521
defun, 1521
kccPage, 1522
  calledby kcaPage, 1522
calledby kcdPage, 1520
calls augmentHasArgs, 1522
calls childrenOf, 1522
calls dbShowCons, 1522
calls errorPage, 1522
calls form2String, 1522
calls htpCopyProplist, 1522
calls htpInitPage, 1522
calls htpProperty, 1522
calls htpSetProperty, 1522
calls kDomainName, 1522
calls opOf, 1522
calls qcar, 1522
calls reduceAlistForDomain, 1522
defun, 1522
kcdePage, 1524
  calls concat, 1524
calls dbShowCons, 1524
calls getConstructorForm, 1524
kcdoPage, 1521
  calledby kcdoPage, 1521
calls domainsOf, 1521
calls kcdoPage, 1521
defun, 1521
kcdPage, 1520
  calls descendantsOf, 1520
calls kcdoPage, 1521
defun, 1520
kCheckArgumentNumber
  calledby kCheckArgumentNumbers, 1531
kCheckArgumentNumbers, 1531
  calledby kisIsValidType, 1530
calls getdatabase, 1531
calls kCheckArgumentNumber, 1531
defun, 1531
kcnPage, 1526
  calledby concat, 1526
calls dbShowCons, 1526
calls errorPage, 1526
calls form2HtString, 1526
calls getImports, 1526
calls htpProperty, 1526
calls htpSetProperty, 1526
calls kDomainName, 1526
calls opOf, 1526
calls pname, 1526
calls qcar, 1526
calls sublistlisa, 1526
defun, 1526
kcPage, 1515
  calledby brCon, 1515
calls dbpHasDefaultCategory?, 1515
calls errorPage, 1515
calls form2HtString, 1515
calls hasNewInfoAlist, 1516
calls hget, 1516
calls htBeginMenu, 1516
calls htCopyProplist, 1515
calls htEndMenu, 1516
calls htInitPage, 1515
calls htMakePage, 1516
calls htSayStandard, 1516
calls htSay, 1515
calls htShowPage, 1516
calls htpProperty, 1515
calls htpSetProperty, 1515
calls kDomainName, 1515
calls nequal, 1516
calls opOf, 1515
calls qcar, 1515
calls satBreak, 1516
uses $defaultPackageNamesHT, 1516
defun, 1515
calledby kcPage, 1515
calledby kcaPage1, 1521
calledby kccPage, 1522
calledby kcnPage, 1526
calledby kcpPage, 1518
calledby kePage, 1509
calledby kiPage, 1508
calledby koPage, 1535
calledby ksPage, 1513
calls concat, 1528
calls dbMkEvalable, 1528
calls getdatabase, 1528
calls htpLabelInputString, 1528
calls htpSetProperty, 1528
calls kArgumentCheck, 1528
calls kIsValidType, 1528
calls mkConform, 1528
calls unabbrev, 1528
uses $PatternVariableList, 1528
catches, 1528
defun, 1528
kcpPage, 1518
calls dbShowCons, 1519
calls errorPage, 1518
calls form2HtString, 1519
calls htCopyProplist, 1519
calls htInitPage, 1519
calls htpProperty, 1515
calls htpSetProperty, 1519
calls kDomainName, 1515
calls opOf, 1519
calls parentsOf, 1519
calls qcar, 1518
calls sublis, 1519
defun, 1518
calledby Skipping?, 306
calledby incLude1, 307
calls remainder, 305
defun, 305
kcuPage, 1525
calls concat, 1525
calls dbShowCons, 1525
calls getConstructorFrom, 1525
calls getcUsersOfConstructor, 1525
calls htpProperty, 1525
calls htpSetProperty, 1525
calls ncPareFromString, 1525
calls nequal, 1525
calls opOf, 1525
defun, 1525
kDomainName, 1528
calledby kPageInfo, 1505
calls bcHt, 1505
calls extractFileNameFromPath, 1505
calls getdatabase, 1505
calls htMakePage, 1505
calls htSayStandard, 1505
calls htSay, 1505
calls kPageArgs, 1505
calls length, 1505
calls nequal, 1505
calls subseq, 1505
defun, 1505
KeepPart?, 305
calledby Skipping?, 306
calledby incLude1, 307
calls remainder, 305
defun, 305
kePage, 1509
calls bcConPredTable, 1510
calls capitalize, 1509
calls concat, 1509
calls errorPage, 1509
calls form2HtString, 1509
calls getConstructorExports, 1510
calls htpBigSkip, 1510
calls htCopyProplist, 1510
calls htInitPage, 1510
calls htMakePage, 1510
calls htSayStandard, 1510
calls htSay, 1510
calls htSowPage, 1510
calls htpProperty, 1509
calls htpSetProperty, 1509, 1510
calls ifcdr, 1510
calls kDomainName, 1509
calls kePageDisplay, 1510
calls kePageOpAlist, 1510
calls length, 1510
calls menuButton, 1510
calls mkConform, 1509
calls opOf, 1509
calls pluralSay, 1510
calls simpHasPred, 1510
calls sublisFormal, 1509
uses $conformsAreDomains, 1510
defun, 1509
kePageDisplay, 1512
calledby dbSpecialExports, 1556
calledby kePage, 1510
calls dbGatherData, 1512
calls dbSowOpItems, 1512
calls htMakePage, 1512
calls htSayStandard, 1512
calls htSay, 1512
calls htpSetProperty, 1512
calls length, 1512
calls menuButton, 1512
calls pluralize, 1512
defun, 1512
kePageOpAlist, 1511
calledby kePage, 1510
calls insertAlist, 1511
calls lassoc, 1511
calls zeroOneConvert, 1511
defun, 1511
keyedSystemError
calledby coerceBranch2Union, 698
calledby compiledLookupCheck, 769
calledby hasAttSig, 656
calledby hasCatExpression, 657
calledby hasCaty1, 662
calledby hasSigAnd, 655
calledby hasSigOr, 656
calledby hasSig, 653
calledby interpret1, 279
calledby pf2Sex1, 531
calledby pfLiteral2Sex, 534
calledby readHiFi, 843
keyword, 344
calledby lfkey, 345
calledby scanCloser?, 349
calledby scanKeyTr, 343
calls hget, 344
defun, 344
keyword?, 344
calledby scanWord, 349
calls hget, 344
defun, 344
kFormatSlotDomain
calledby dbAddChainDomain, 1544
calledby dbSearchOrder, 1514
kiPage, 1508
calls capitalize, 1509
calls dbShowConsDoc1, 1509
calls errorPage, 1509
calls htCopyProplist, 1509
calls htInitPage, 1509
calls htShowPage, 1509
calls htpProperty, 1508
calls kDomainName, 1508
calls mkConform, 1508
uses $conformsAreDomains, 1509
defun, 1508
kisValidType, 1530
calledby consSpecialString?, 1502
calledby kDomainName, 1528
calls kCheckArgumentNumbers, 1530
calls member, 1530
calls processInteractive, 1530
uses $ProcessInteractiveValue, 1530
uses $noEvalTypeMsg, 1530
catches, 1530
defun, 1530
knownEqualPred
calledby hashable, 1246, 1247
koaPageFilterByCategory
calledby koaPageFilterByName, 1537
koPageFilterByName, 1537
  calls dbGetInputString, 1538
  calls downcase, 1538
  calls htpLabelInputString, 1537
  calls htpProperty, 1538
  calls htpSetProperty, 1538
  calls koaPageFilterByCategory, 1537
  calls pmTransFilter, 1538
  calls superMatch?, 1538
  defun, 1537
koAttrs
  calledby koPageAux, 1537
koOps
  calledby koPageAux, 1537
koPage, 1535
  calledby conOpPage1, 1533
  calls capitalize, 1535
  calls concat, 1535
  calls errorPage, 1535
  calls form2HtString, 1535
  calls htpProperty, 1535
  calls htpSetProperty, 1535
  calls kDomainName, 1535
  calls koPageAux, 1535
  calls koPageInputAreaUnchanged?, 1535
  defun, 1535
koPageAux, 1536
  calledby koPageFromKKPage, 1536
  calledby koPage, 1535
  calls assoc, 1537
  calls dbShowOperationsFromConform, 1537
  calls htpSetProperty, 1536
  calls koAttrs, 1537
  calls koOps, 1537
  calls systemError, 1537
  defun, 1536
koPageAux1, 1537
  calledby dbShowOperationsFromConform, 1537
  calls htpProperty, 1537
  defun, 1537
koPageFromKKPage, 1536
  calls htpProperty, 1536
  calls koPageAux, 1536
  defun, 1536
koPageInputAreaUnchanged?, 1527
  calledby koPage, 1535
  calls concat, 1527
  calls htpLabelInputString, 1527
  calls htpProperty, 1527
  defun, 1527
kPage
  calledby conPage, 1503
  calledby constructorSearch, 1501
  calls addParameterTemplates, 1497
  calls capitalize, 1497
  calls dbConformGenUnder, 1497
  calls dbShowConsDocl, 1497
  calls dbSourceFile, 1497
  calls dbXParts, 1497
  calls htAddHeading, 1497
  calls htInitPageNoScroll, 1497
  calls htSayStandard, 1497
  calls htSay, 1497
  calls htShowPageNoScroll, 1497
  calls htpSetProperty, 1497
  calls isExposedConstructor, 1497
  calls kPageContextMenu, 1497
  calls mkConform, 1497
  calls ncParseFromString, 1497
  calls opOf, 1497
  calls strconc, 1497
  uses $atLeastOneUnexposed, 1497
  uses $conformsAreDomains, 1497
kPageArgs
  calledby kdPageInfo, 1505
kPageContextMenu
  calledby kPage, 1497
kSearch, 1500
  calledby constructorSearch, 1500
  defun, 1500
ksPage, 1513
  calledby dbSearchOrder, 1513
  calledby dbShowCons, 1513
  calledby errorPage, 1513
  calledby form2HtString, 1513
  calledby htCopyProplist, 1513
  calledby htInitPageNoScroll, 1513
  calledby htSayStandard, 1513
  calledby htSay, 1513
  calledby htpProperty, 1513
  calledby htpSetProperty, 1513
  calls kDomainName, 1513
defun, 1513
kTestPred, 1543
called by dbSearchOrder, 1514
calls simpHasPred, 1543
calls testBitVector, 1543
uses $domain, 1543
uses $predvec, 1543
defun, 1543
lassoc
called by bcMatrixGen, 1341
called by breaklet, 86
called by coerceTraceFunValue2E, 64
called by conOpPage1, 1533
called by constructorSearch, 1501
called by genDomainTraceName, 59
called by getConstantFromDomain, 699
called by getConstructorDocumentation, 1551
called by kePageOpAlist, 1511
called by letPrint2, 1064
called by letPrint3, 1066
called by letPrint, 1062
called by processSynonyms, 257
called by reportUndo, 44
called by serverReadLine, 269
called by set1, 1014
called by subTypes, 1055
called by trace1, 48
called by tracelet, 85
called by undoSingleStep, 39
called by unifyStructVar, 659
lassocSub, 1057
called by untrace, 59
calls lassq, 1057
defun, 1057
lassq
called by conPageFastPath, 1504
called by conPage, 1503
called by diffAlist, 1085
called by lassocSub, 1057
last
called by coerceIntTower, 682
called by frameSpad2Cmd, 11
lastatom
called by dbGetDocTable,gn, 1542
lastc
called by StringToDir, 1228
lastcount, 888
called by startp, 887
defun, 888
lastTokPosn, 563
called by enPile, 562
called by separatePiles, 563
calls tokPosn, 563
defun, 563
leader?, 573
called by msgOutputter, 576
called by showMsgPos?, 580
calls getMsgTag, 573
defmacro, 573
leave, 620
syntax, 620
leaveScratchpad, 873
called by intloopReadConsole, 253
called by quitSpad2Cmd, 872
defun, 873
lefts, 1562
calls hkeys, 1562
calls hascategory-hash, 1562
defun, 1562
length
called by augmentHasArgs, 1523
called by bcDifferentiateGen, 1350
called by bcInputExplicitMatrix, 1339
called by coerceIntByMap, 693
called by computeTTTranspositions, 686
called by conPageFastPath, 1504
called by dbConsHeading, 1553
called by dbConstructorDoc,lm, 1538
called by dbShowConditions, 1552
called by kdPageInfo, 1505
called by kePageDisplay, 1512
called by kePage, 1510
called by libConstructorSig, 1563
called by mkDomPvar, 663
letPrint, 1062
called by mapLetPrint, 74
calls break, 1062
calls bright, 1062
calls gensymp, 1062
calls hasPair, 1062
calls isSharpVarWithNum, 1062
calls isgenvar, 1062
calls lassoc, 1062
calls pname, 1062
calls sayBrightlyNT, 1062
defun, 1062
letPrint2, 1064
calls break, 1065
calls bright, 1065
calls gensymp, 1064
calls hasPair, 1065
calls isSharpVarWithNum, 1064
calls isgenvar, 1064
calls lassoc, 1064
calls mathprint, 1064
calls pname, 1065
calls print, 1065
uses $BreakMode, 1065
uses $letAssoc, 1065
defun, 1064
letPrint3, 1066
calls break, 1066
calls bright, 1066
calls gensymp, 1066
calls hasPair, 1066
calls isSharpVarWithNum, 1066
calls isgenvar, 1066
calls lassoc, 1066
calls mathprint, 1066
calls pname, 1066
calls print, 1066
calls spadcall, 1066
uses $BreakMode, 1066
uses $letAssoc, 1066
catches, 1066
defun, 1066
lfcomment, 340
calledby scanComment, 339
defun, 340
lferror, 359
calledby scanError, 358
defun, 359
lffloat, 351
calledby scanExponent, 349
calls concat, 351
defun, 351
lfid, 338

calledby scanToken, 337
calledby scanWord, 349
defun, 338
lfinteger, 357
calledby scanNumber, 355
defun, 357
lfkey, 345

calledby scanKeyTr, 343
calledby scanPossFloat, 344
calledby scanWord, 349
calls keyword, 345
defun, 345
lfnegcomment, 341
calledby scanNegComment, 340
defun, 341
lfinteger, 357
calledby scanNumber, 355
calls concat, 357
defun, 357
lfspaces, 353

calledby scanSpace, 352
defun, 353
lfstring, 353

calledby scanString, 353
defun, 353
libConstructorSig, 1563
calls form2LispString, 1563
calls getdatabase, 1563
calls length, 1563
calls ncParseFromString, 1563
calls sayBrightly, 1563
calls sublis, 1563
calls take, 1563
uses $TriangleVariableList, 1563
defun, 1563
libdbTrim, 1500
defun, 1500
library, 864, 1159
calls extendLocalLibdb, 1159
calls localdatabase, 1159
calls tersyscommand, 1159
uses $newConlist, 1159
uses $options, 1159
defun, 1159
manpage, 864
libstream, 1314
defstruct, 1314
license, 866
calls concat, 866
calls getenv, 866
calls obey, 866
manpage, 866
line
usedby commandErrorIfAmbiguous, 742
usedby commandErrorMessage, 722
usedby doSystemCommand, 718
usedby intProcessSynonym, 257
usedby processSynonym, 257
usedby unAbbreviateKeyword, 741
line-clear
calledby ioclear, 1118
line-handler, 1113
usedby init-boot/spad-reader, 1114
defvar, 1113
line-past-end-p
calledby iostat, 1116
calledby spad-short-error, 1116
line-print
calledby iostat, 1116
calledby spad-short-error, 1116
line?, 573
calledby msgOutputter, 576
calls getMsgTag, 573
defmacro, 573
linearFinalRequest, 1395
calls bcQueryInteger, 1395
calls explainLinear, 1395
calls sayBrightly, 1395
defun, 1395
linearFormatName
calledby sayCacheCount, 915
lineoftoks, 334
calledby intloopInclude0, 288
calledby nclloopInclude0, 297
calledby parseFromstring, 273
calledby parseNoMacroFromString, 1531
calls constoken, 335
calls dqUnit, 335
calls incPrefix?, 335
calls nextline, 335
calls scanIgnoreLine, 335
calls substring, 335
uses $floatok, 335
uses $f, 335
uses $linepos, 335
uses $ln, 335
uses $n, 335
uses $r, 335
uses $sz, 335
defun, 334
lisp, 867
manpage, 867
list2vec
calledby computeTTTranspositions, 686
listConstructorAbbreviations, 756
calledby abbreviationsSpad2Cmd, 754
calledby displaySpad2Cmd, 796
calls queryUserKeyedMsg, 756
calls sayKeyedMsg, 756
calls string2id-n, 756
calls upcase, 756
calls whatSpad2Cmd, 756
defun, 756
listDecideHowMuch, 583
calledby getPosStL, 578
calls poGlobalLinePosn, 583
calls poNopos?, 583
calls poPosImmediate?, 583
defun, 583
ListMember?
calledby whichCat, 587
listOfStrings2String, 1454
defun, 1454
listOutputter, 576
calledby processMsgList, 591
calls msgOutputter, 576
defun, 576
listSort
calledby dbShowCons1, 1547
calledby dbShowConstructorLines, 1554
calledby domainDescendantsOf, 1507
calledby erMsgSort, 591
calledby kcaPage1, 1521
INDEX

literals
called by displaySetOptionInformation, 896
loadLib, 1181
called by displaySetVariableSettings, 897
called by initializeSetVariables, 894
called by set1, 1015
lnCreate, 567
called by incLine1, 313
defun, 567
lnExtraBlanks, 567
called by scanError, 358
called by scan$S, 354
called by scanToken, 338
defun, 567
lnFileName, 569
called by poFileName, 582
calls lnFileName?, 569
calls ncBug, 569
defun, 569
lnFileName?, 569
called by lnFileName, 569
called by lnImmediate?, 569
defun, 569
lnGlobalNum, 568
called by poGlobalLinePosn, 296
defun, 568
lnImmediate?, 569
called by poPosImmediate?, 582
calls lnFileName?, 569
defun, 569
lnLocalNum, 568
called by poLinePosn, 583
defun, 568
lnPlaceOfOrigin, 568
defun, 568
lnSetGlobalNum, 568
called by incRenumberItem, 300
defun, 568
lnString, 567
defun, 567
loadFunctor, 1184
called by loadFunctor, 1184
called by traceDomainConstructor, 71
calls loadFunctor, 1184
calls loadLibIfNotLoaded, 1184
defun, 1184
loadIfNecessary
called by mkEvalable, 1076
called by browse, 762
calls categoryForm?, 1182
calls clearConstructorCache, 1181
calls getDatabase, 1181
calls installConstructor, 1181
calls isSystemDirectory, 1181
calls loadLibNoUpdate, 1181
calls namestring, 1181
calls pathnameDirectory, 1181
calls remprop, 1182
calls sayKeyedMsg, 1181
calls startTimingProcess, 1181
calls stopTimingProcess, 1182
calls updateCategoryTable, 1182
calls updateDatabase, 1181
calls sayKeyedMsg, 1181
calls getDatabase, 1183
calls updateDatabase, 1183
local def $CategoryFrame, 1182
local ref $InteractiveMode, 1182
local ref $forceDatabaseUpdate, 1182
local ref $printLoadMsgs, 1182
defun, 1181
loadLibIfNotLoaded
called by loadFunctor, 1184
loadLibNoUpdate, 1183
called by loadLib, 1181
called by localnrlib, 1161
calls clearConstructorCache, 1183
calls getDatabase, 1183
calls installConstructor, 1183
calls sayKeyedMsg, 1183
calls stopTimingProcess, 1183
calls toplevel, 1183
calls sayKeyedMsg, 1183
calls $InteractiveMode, 1183
local ref $forceDatabaseUpdate, 1183
defun, 1183
localdatabase, 1160
called by library, 1159
called by make-databases, 1165
calls localnrlib, 1160
calls sayKeyedMsg, 1160
globules *index-filename*, 1160
globules $ConstructorCache, 1160
globules $forceDatabaseUpdate, 1160
defun, 1160
localrlib, 1161
called by localdatabase, 1160
calls addoperations, 1161
calls categoryForm?, 1161
calls getdatabase, 1161
calls installConstructor, 1161
calls loadLibNoUpdate, 1161
calls make-database, 1161
calls sayKeyedMsg, 1161
calls installConstructor, 1161
calls startTimingProcess, 1161
calls sublislis, 1161
calls updateCategoryTable, 1161
calls updateDatabase, 1161
calls *allOperations*, 1161
calls *allconstructors*, 1161
calls $FormalMapVariableList, 1161
defun, 1161
localVars, 240
defvar, 240
lookupInDomainVector, 1187
called by basicLookup, 1186
called by oldCompLookup, 1189
calls basicLookupCheckDefaults, 1187
calls spadcall, 1187
defun, 1187
loopIters2Sex, 541
called by pf2Sex1, 530
called by pfCollect2Sex, 544
calls pf2Sex1, 542
defun, 541
lotsof, 1194
called by wrap, 1193
defun, 1194
ltrace, 868
calls trace, 868
defun, 868
manpage, 868
mac0Define, 458
called by mac0MLambdaApply, 452
called by macMacro, 458
uses $pfMacros, 458
defun, 458
mac0ExpandBody, 453
called by mac0MLambdaApply, 452
called by macId, 455
calls mac0InfiniteExpansion, 453
calls macExpand, 453
calls pfSourcePosition, 453
uses $macActive, 453
uses $posActive, 453
defun, 453
mac0Get, 456
called by macId, 455
calls ifcdr, 456
uses $pfMacros, 456
defun, 456
mac0GetName, 454
called by mac0InfiniteExpansion, 454
calls pfMLambdaBody, 454
uses $pfMacros, 454
defun, 454
mac0InfiniteExpansion, 453
called by mac0ExpandBody, 453
calls mac0InfiniteExpansion, 453
calls ncSoftError, 453
calls pform, 453
defun, 453
mac0InfiniteExpansion, 454
called by mac0InfiniteExpansion, 454
calls mac0GetName, 454
calls pname, 454
defun, 454
mac0MLambdaApply, 452
called by macApplication, 451
calls mac0Define, 452
calls mac0ExpandBody, 452
calls ncHardError, 452
calls pfMLambdaArgs, 452
calls pfId?, 452
calls pfMLambdaBody, 452
calls pfSourcePosition, 452
calls pform, 452
uses $macActive, 452
uses $pfMacros, 452
uses $posActive, 452
defun, 452
mac0SubstituteOuter, 459
called by mac0SubstituteOuter, 459
called by macSubstituteOuter, 459
calls mac0SubstituteOuter, 459
calls macLambdaParameterHandling, 459
calls macSubstituteId, 459
calls pfId?, 459
calls pfLambda?, 459
calls pfLeaf?, 459
calls pfParts, 459
defun, 459
macApplication, 451
called by macExpand, 451
calls mac0MLambdaApply, 451
calls macExpand, 451
calls pf0ApplicationArgs, 451
calls pfMLambda?, 451
calls pfMapParts, 451
uses $pfMacros, 451
defun, 451
macExpand, 450
called by mac0ExpandBody, 453
called by macApplication, 451
called by macExpand, 451
called by macLambda, 457
called by macWhere, 457
called by macroExpanded, 450
calls macApplication, 451
calls macExpand, 451
calls macId, 451
calls macLambda, 450
calls macMacro, 451
calls macWhere, 450
calls pfApplication?, 451
calls pfId?, 451
calls pfLambda?, 450
calls pfMacro?, 451
calls pfMapParts, 451
calls pfWhere?, 450
defun, 450
macId, 455
called by macExpand, 451
calls mac0ExpandBody, 455
calls mac0Get, 455
calls pfCopyWithPos, 455
calls pfIdSymbol, 455
calls pfSourcePosition, 455
uses $macActive, 455
uses $posActive, 455
defun, 455
macLambda, 457
called by macExpand, 450
calls macLambda, 450
uses $pfMacros, 457
defun, 457
macLambda, 457
called by macLambda, 457
calls macExpand, 457
calls pfMapParts, 457
uses $pfMacros, 457
defun, 457
macLambdaParameterHandling, 459
called by mac0SubstituteOuter, 459
called by macLambdaParameterHandling, 460
called by mac0SubstituteOuter, 459
called by macLambdaParameterHandling, 460
calls pf0LambdaArgs, 460
calls pf0MLambdaArgs, 460
calls pfAbSynOp, 460
calls pfIdSymbol, 460
calls pfLambda?, 460
calls pfLeaf?, 459
calls pfLeafPosition, 460
calls pfLeaf, 460
calls pfMLambda?, 460
calls pfParts, 460
calls pfTypedId, 460
defun, 459
macMacro, 457
called by macExpand, 451
calls mac0Define, 458
calls macSubstituteOuter, 458
calls ncSoftError, 458
calls pfId?, 458
calls pfIdSymbol, 458
calls pfMLambda?, 458
calls pfMacroLhs, 457
calls pfMacroRhs, 458
calls pfMacro, 458
calls pfNothing?, 458
calls pfNothing, 458
calls pfSourcePosition, 458
calls pfForm, 458
defun, 457
macroExpanded, 450
called by parseFromString, 273
called by phMacro, 450
calls macroExpand, 450
uses $macActive, 450
uses $posActive, 450
defun, 450
macSubstituteId, 460
called by macroSubstituteOuter, 459
calls pfIdSymbol, 460
defun, 460
macSubstituteOuter, 459
called by macroMacro, 458
calls macroSubstituteOuter, 459
calls macroLambdaParameterHandling, 459
defun, 459
macWhere, 456
called by macroExpand, 450
calls macroWhere, 456
uses $pfMacros, 456
defun, 456
macWhere, 456
called by macroWhere, 456
calls macroExpand, 457
calls pfMapParts, 456
uses $pfMacros, 457
defun, 456
make-absolute-filename, 260
called by reroot, 265
uses $spadroot, 260
defun, 260
make-appendstream, 1128
called by makeStream, 1129
calls make-filename, 1128
defun, 1128
make-cdouble-matrix, 1207
defmacro, 1207
make-cdouble-vector, 1208
defmacro, 1208
make-database

called by interpopen, 1150
called by localnrlib, 1161
called by setdatabase, 1155
make-databases, 1164
calls allConstructors, 1165
calls browserAutoloadOnceTrigger, 1165
calls buildLibdb, 1165
calls categoryForm?, 1165
calls dsSplitLibdb, 1165
calls domainsOf, 1165
calls getConstructorForm, 1165
calls getenvir, 1165
calls localdatabase, 1165
calls mkDependentsHashTable, 1165
calls mkTopicHashTable, 1165
calls mkUsersHashTable, 1165
calls saveDependentsHashTable, 1165
calls saveUsersHashTable, 1165
calls write-browsedb, 1165
calls write-categorydb, 1165
calls write-interpdb, 1165
calls write-operationdb, 1165
calls write-warmdata, 1165
uses *allconstructors*, 1165
uses *operation-hash*, 1165
uses *sourcefiles*, 1165
uses $constructorList, 1165
defun, 1164
make-double-matrix, 1224
defmacro, 1224
make-double-matrix1, 1225
defmacro, 1225
make-double-vector, 1226
defmacro, 1226
make-double-vector1, 1226
defmacro, 1226
make-filename

called by make-appendstream, 1128
called by make-outstream, 1127
called by pathname, 1192
make hashtable
called by dbDocTable, 1539
make-instream, 1127
called by dewritify,dewritifyInner, 854
called by incConsoleInput, 328
called by incFileInput, 327
called by newHelpSpad2Cmd, 813
calls makeInputFilename, 1127
defun, 1127
make-monitor-data

called by monitor-add, 1316
make-outstream, 1127
INDEX

1747

called by makeStream, 1129
called by setOutputAlgebra, 967
called by setOutputFormula, 995
called by setOutputHtml, 980
called by setOutputMathml, 986
called by setOutputOpenMath, 991
called by setOutputTex, 1002
calls make-filename, 1127
defun, 1127

make-string
called by executeQuietCommand, 272
makeByteWordVec2, 1305
defun, 1305
makeFullNamestring, 1131
called by makeStream, 1131
defun, 1131

makeHistFileName, 819
called by closeInterpreterFrame, 22
called by histFileName, 820
called by oldHistFileName, 819
called by restoreHistory, 839
called by saveHistory, 837
calls makePathname, 819
defun, 819

makeInitialModemapFrame, 261
called by restart, 239
calls copy, 261
uses $InitialModemapFrame, 261
defun, 261

makeInputFilename, 1129
called by initHist, 820
called by make-instream, 1127
called by newHelpSpad2Cmd, 813
called by restoreHistory, 839
called by saveHistory, 837
called by updateSourceFiles, 806
called by workfilesSpad2Cmd, 1099
defun, 1129

makeLeaderMsg, 599
called by processChPosesForOneLine, 598
uses $nupos, 599
uses $preLength, 599
defun, 599

makeLongSpaceString
called by printStorage, 283

makeLongTimeString
called by printTypeAndTime, 284

makeMatrix1U16, 1253
defmacro, 1253
makeMatrix1U32, 1254
defmacro, 1254
makeMatrix1U8, 1251
defmacro, 1251
makeMatrixU16, 1252
defmacro, 1252
makeMatrixU32, 1254
defmacro, 1254
makeMatrixU8, 1251
defmacro, 1251

makeMsgFromLine, 593
called by processMsgList, 591
calls char, 593
calls getLinePos, 593
calls getLineText, 593
calls poGlobalLinePosn, 593
calls poLinePosn, 593
calls rep, 593
calls size, 593
calls strconc, 593
uses $preLength, 593
defun, 593

makeOrdinal, 1083
called by evaluateType1, 1081
defun, 1083

makePathname, 1193
called by histInputFileName, 820
called by makeHistFileName, 819
called by readSpad2Cmd, 875
called by sayMSG2File, 28
calls object2String, 1193
calls pathname, 1193
defun, 1193

makeSpadCommand, 1448
defun, 1448

makeStream, 1129
called by setOutputFortran, 974
calls make-appendstream, 1129
calls make-outstream, 1129
calls make-filename, 1129
defun, 1129

MakeSymbol
called by assertCond, 321
INDEX

called by ifCond, 314
manexp, 1220
defun, 1220
mapage
abbreviations, 753
boot, 758
browse, 759
cd, 764
clear, 765
close, 776
compile, 779
copyright, 782
credits, 788
describe, 789
display, 794
edit, 803
fin, 808
frame, 3
help, 810
history, 815
include, 861
library, 864
license, 866
lisp, 867
ltrace, 868
pquit, 869
quit, 871
read, 874
regress, 878
savesystem, 890
set, 892
show, 1018
spool, 1033
summary, 1034
synonym, 1036
system, 1041
tangle, 1042
trace, 1045
trademark, 1044
undo, 1072
what, 1089
workfiles, 1099
mapLetPrint, 74
calls getAliasIfTracedMapParameter, 74
calls getBpiNameIfTracedMap, 74
calls letPrint, 74
defun, 74
mapStringize, 1414
defun, 1414
mathprint
called by displayMacro, 726
called by displayValue, 731
called by letPrint2, 1064
called by letPrint3, 1066
maxindex
called by dbWordFrom, 1496
called by processSynonymLine, removeKeyFromLine, 1039
called by removeUndoLines, 41
member, 1198
called by bcSolveEquations, 1390
called by clearCmdParts, 773
called by coerceIntByMap, 693
called by coerceIntPermute, 685
called by computeTTTranspositions, 686
called by conSpecialString?, 1502
called by dbConsHeading, 1553
called by dbShowConsDoc1, 1550
called by dbShowCons, 1545
called by dbWordFrom, 1496
called by displayMacros, 799
called by displayProperties, 733
called by doSystemCommand, 718
called by evaluateType, 1078
called by fixObjectForPrinting, 728
called by getBrowseDatabase, 1241
called by handleNoParseCommands, 742
called by hasCateSpecialNew, 666
called by historySpad2Cmd, 822
called by importFromFrame, 19
called by interpret2, 280
called by isDomainValuedVariable, 1105
called by isSubForRedundantMapName, 1059
called by isValidType, 1530
called by readSpad2Cmd, 875
called by reportOpsFromUnitDirectly, 1028
called by retract2Specialization, 704
called by selectOption, 751
called by setExposeAddConstr, 92
called by setExposeAddGroup, 89
called by setExposeDropConstr, 96
called by setExposeDropGroup, 95
INDEX

called by setOutputAlgebra, 967
called by setOutputFormula, 995
called by setOutputFortran, 974
called by setOutputHtml, 980
called by setOutputMathml, 986
called by setOutputOpenMath, 990
called by setOutputTex, 1002
called by showSpad2Cmd, 1019
called by transTraceItem, 60
called by translateYesNo2TrueFalse, 899
called by updateSourceFiles, 806
defun, 1198
menuButton
called by kePageDisplay, 1512
called by kePage, 1510
mergePathnames, 1191
called by readSpad2Cmd, 875
calls pathnameDirectory, 1191
calls pathnameName, 1191
calls pathnameType, 1191
defun, 1191
messageprint, 1199
called by brightprint, 1198
defun, 1199
messageprint-1, 1199
called by brightprint-0, 1198
called by messageprint-1, 1199
called by messageprint-2, 1199
calls identp, 1199
calls messageprint-1, 1199
calls messageprint-2, 1199
defun, 1199
messageprint-2, 1199
called by messageprint-1, 1199
called by messageprint-2, 1199
calls messageprint-1, 1199
calls messageprint-2, 1199
defun, 1199
meta-error-handler
used by init-boot/spad-reader, 1114
mkAtree
called by coerceInt1, 675
called by evaluateType1, 1080
called by evaluateType, 1078
called by getSubDomainPredicate, 701
called by reportOperations, 1021
mkAtreeNode
called by coerceInt1, 675
mkAtreeWithSrcPos
called by interpret1, 279
mkCompanionPage
called by recordAndPrint, 282
mkConform, 1532
called by conOpPage1, 1533
called by kDomainName, 1528
called by kPage, 1497
called by kePage, 1509
called by kiPage, 1508
calls concat, 1532
calls ncParseFromString, 1532
calls nequal, 1532
calls parseNoMacroFromString, 1532
calls pp, 1532
calls sayBrightlyNT, 1532
calls systemError, 1532
defun, 1532
mkCurryFun, 1425
defun, 1425
mkDependentsHashTable
called by make-databases, 1165
mkDomPvar, 663
called by hasCaty, 650
calls domArg, 663
calls length, 663
local ref $FormalMapVariableList, 663
defun, 663
mkDomTypeForm, 1507
called by kArgPage, 1506
called by mkDomTypeForm, 1507
calls hasIndent, 1507
calls mkDomTypeForm, 1507
calls sublislis, 1507
defun, 1507
mkEvalable, 1075
called by dbMkEvalable, 1529
called by evalDomain, 1075
called by mkEvalableMapping, 1078
called by mkEvalableRecord, 1078
called by mkEvalableUnion, 1077
called by mkEvalable, 1076
called by writify,writifyInner, 849
calls bpiname, 1076
calls constructor?, 1076
calls devaluate, 1076
calls fbpip, 1076
calls getdatabase, 1076
calls loadIfNecessary, 1076
calls mkEvalableMapping, 1076
calls mkEvalableRecord, 1076
calls mkEvalableUnion, 1076
calls mkEvalable, 1076
calls mkq, 1076
calls qcar, 1075
calls qcdr, 1075
local ref $EmptyMode, 1076
local ref $Integer, 1076
defun, 1075
mkEvalableMapping, 1078
calledby mkEvalable, 1076
calls mkEvalable, 1078
defun, 1078
mkEvalableRecord, 1078
calledby mkEvalable, 1076
calls mkEvalable, 1078
defun, 1078
mkEvalableUnion, 1077
calledby mkEvalable, 1076
calls mkEvalable, 1077
defun, 1077
mkLineList, 295
calledby intloopEchoParse, 294
defun, 295
mkObj, 446
calledby coerceBranch2Union, 698
calledby coerceByTable, 691
calledby coerceInt0, 674
calledby coerceInt1, 675
calledby coerceInteractive, 672
calledby coerceUnion2Branch, 707
calledby interpret1, 279
calledby interpret2, 280
calledby retract2Specialization, 704
calledby retract, 1203
defmacro, 446
mkObjCode, 447
defmacro, 447
mkobjFn, 1237
defun, 1237
mkObjWrap, 447
calledby coerceBranch2Union, 698
calledby coerceByTable, 691
calledby coerceInt1, 675
calledby coerceIntAlgebraicConstant, 698
calledby coerceIntByMap, 693
calledby coerceIntCommute, 689
calledby coerceIntX, 700
calledby coerceInteractive, 672
calledby coerceOrThrowFailure, 695
calledby coerceRetract, 709
calledby coerceSpadArgs2E, 63
calledby coerceSpadFunValue2E, 64
calledby coerceTraceArg2E, 62
calledby coerceTraceFunValue2E, 64
calledby getAndEvalConstructorArgument, 1104
calledby printTypeAndTime, 284
calledby recordAndPrint, 282
calledby retract2Specialization, 704
calledby retractByFunction, 710
calledby valueArgsEqual?, 696
calls wrap, 447
defmacro, 447
mkPredList
calledby coerceBranch2Union, 698
calledby coerceUnion2Branch, 707
mkprompt, 266
calledby SpadInterpretStream, 252
calledby intloopReadConsole, 253
calledby serverReadLine, 269
calls concat, 266
calls substring, 267
uses $IOindex, 267
uses $inputPromptType, 267
uses $InterpreterFrameName, 267
defun, 266
mkq
calledby coerceByTable, 691
calledby mkEvalable, 1076
calledby traceDomainConstructor, 71
mkSetTitle, 1454
defun, 1454
mkTopicHashTable
calledby make-databases, 1165
mkUnixPattern, 1492
defun, 1492
mkUserConstructorAbbreviation
called by abbreviationsSpad2Cmd, 755
mkUsersHashTable
called by make-databases, 1165
modes
called by clearCmdParts, 773
MONITOR,EVALTRAN
called by break, 87
monitor-add, 1316
called by monitor-file, 1319
calls make-monitor-data, 1316
calls monitor-delete, 1316
uses *monitor-table*, 1316
defun, 1316
monitor-apropos, 1329
uses *monitor-table*, 1329
defun, 1329
monitor-autoload, 1324
defun, 1324
monitor-checkpoint, 1321
uses *monitor-table*, 1321
uses *print-package*, 1321
defun, 1321
monitor-data, 1314
defstruct, 1314
monitor-decr, 1318
uses *monitor-table*, 1318
defun, 1318
monitor-delete, 1316
called by monitor-add, 1316
called by monitor-tested, 1320
uses *monitor-table*, 1316
defun, 1316
monitor-dirmname, 1324
uses *monitor-nrlibs*, 1324
defun, 1324
monitor-disable, 1317
uses *monitor-table*, 1317
defun, 1317
monitor-enable, 1316
uses *monitor-table*, 1316
defun, 1316
monitor-end, 1315
uses *monitor-table*, 1315
defun, 1315
monitor-exposedp, 1326
monitor-file, 1319
calls monitor-add, 1319
catches, 1319
defun, 1319
throws, 1319
monitor-help, 1322
monitor-incr, 1318
uses *monitor-table*, 1318
defun, 1318
monitor-info, 1319
uses *monitor-table*, 1319
defun, 1319
monitor-initable, 1314
uses *monitor-table*, 1314
defun, 1314
monitor-libname, 1325
defun, 1325
monitor-parse, 1328
called by monitor-spadfile, 1328
defun, 1328
monitor-percent, 1329
uses *monitor-table*, 1329
defun, 1329
monitor-readinterp, 1326
called by monitor-report, 1327
uses *monitor-domains*, 1326
catches, 1326
defun, 1326
throws, 1326
monitor-report, 1327
called by monitor-readinterp, 1327
uses *monitor-domains*, 1327
defun, 1327
monitor-reset, 1317
uses *monitor-table*, 1317
defun, 1317
monitor-restore, 1321
defun, 1321
monitor-results, 1315
uses *monitor-table*, 1315
defun, 1315
monitor-spadfile, 1328
calls monitor-parse, 1328
uses *monitor-domains*, 1328
catches, 1328
defun, 1328
throws, 1328
monitor-tested, 1320
calls monitor-delete, 1320
uses *monitor-table*, 1320
defun, 1320
monitor-untested, 1319
uses *monitor-table*, 1319
defun, 1319
monitor-write, 1320
defun, 1320
moreExactSolution

called by finalExactRequest, 1396
msgCreate, 569
called by ncBug, 590
called by ncHardError, 574
called by ncSoftError, 574
calls initImPr, 570
calls initToWhere, 570
calls putDatabaseStuff, 570
calls setMsgForcedAttrList, 570
defun, 569
msgImPr?, 580
called by alreadyOpened?, 585
called by getPosStL, 578
called by processKeyedError, 575
called by showMsgPos?, 580
calls getMsgCatAttr, 580
defun, 580
msgNoRep?, 597
called by redundant, 596
calls getMsgCatAttr, 597
defun, 597
msgOutputter, 575
called by listOutputter, 576
called by processKeyedError, 575
calls alreadyOpened?, 576
calls flowSegmentedMsg, 576
calls getStFromMsg, 575
calls leader?, 576
calls line?, 576
calls sayBrightly, 576
calls toFile?, 576
calls toScreen?, 576
uses $linelength, 576
defun, 575
msgText, 286
called by printTypeAndTime, 284
calls flowSegmentedMsg, 286
calls segmentKeyedMsg, 286
calls substituteSegmentedMsg, 286
uses $linelength, 286
uses $margin, 286
defun, 286
msort
called by apropos, 1097
called by computeTTTranspositions, 686
called by displayOperationsFromLisplib, 1027
called by displayProperties, 733
called by displayWorkspaceNames, 727
called by reportOpsFromLisplib, 1025
called by reportOpsFromUnitDirectly, 1028
called by saveDependentsHashTable, 1168
called by saveUsersHashTable, 1168
called by setExposeAddConstr, 93
called by setExposeAddGroup, 89
called by setExposeDropConstr, 96
called by whatConstructors, 1097
msubst
called by getSubDomainPredicate, 701
myWritable?, 1231
called by error, 1231
called by frameDirectory, 1231
calls frameExists?, 1231
calls writeablep, 1231
defun, 1231
myWriteable?
called by fnameWriteable?, 1230
names
called by frameSpad2Cmd, 11
namestring, 1190
called by bcUnixTable, 1554
called by editFile, 805
called by loadLib, 1181
called by newHelpSpad2Cmd, 813
called by readSpad2Cmd, 875
called by reportOpsFromLispLib, 1024
called by reportOpsFromUnitDirectly, 1028
called by restoreHistory, 839
called by saveHistory, 837
called by setExposeAddGroup, 89
called by setExpose, 91
called by workfilesSpad2Cmd, 1099
called by writeInputLines, 828
calls pathname, 1190
defun, 1190
ncAbort
  called by ncBug, 590
ncAlist, 637
called by getMsgCatAttr, 581
called by ncElq, 638
called by ncPutq, 639
called by setMsgCatAddAttr, 587
called by setMsgUnforcedAttr, 589
called by tokPosn, 635
calls identp, 637
calls qcar, 637
calls qcdr, 638
defun, 637
ncBug, 590
called by getMsgPos2, 601
called by incHandleMessage, 301
called by lnFileName, 569
called by ncallist, 637
called by ncElq, 638
called by ncTag, 637
called by poGlobalLinePosn, 296
calls enable-backtrace, 590
calls msgCreate, 590
calls ncAbort, 590
calls processKeyedError, 590
uses $newcompErrorCount, 590
uses $nopos, 590
defun, 590
ncConversationPhase, 292
called by intloopSpadProcess, interp, 290
called by ncConversationPhase, wrapup, 293
called by ncHardError, 574
uses $ncMsgList, 293
defun, 293
ncElq, 638
called by intloopSpadProcess, interp, 290
called by phIntReportMsgs, 291
called by phInterpret, 292
called by phMacro, 450
calls ncAlist, 638
calls ncBug, 638
calls qassq, 638
defun, 638
ncError, 293
called by intloopSpadProcess, interp, 290
called by ncHardError, 574
defun, 293
throws, 293
ncHardError, 574
called by mac0MLambdaApply, 452
calls desiredMsg, 574
calls msgCreate, 574
calls ncError, 574
calls processKeyedError, 574
uses $newcompErrorCount, 574
defun, 574
ncIntLoop, 249
called by ncTopLevel, 248
called by intloop, 249
uses curinstream, 249
uses curoutstream, 249
defun, 249
ncloopCommand, 750
called by intloopReadConsole, 253
calls $systemCommandFunction, 750
calls ncloopInclude1, 750
calls ncloopPrefix?, 750
uses $systemCommandFunction, 750
defun, 750
ncloopDQlines, 296
called by intloopEchoParse, 294
called by ncloopParse, 262
calls StreamNull, 296
calls poGlobalLinePosn, 296
calls streamChop, 296
calls tokPosn, 296
defun, 296
ncloopEchoParse
called by ncloopInclude0, 297
cncloopEscaped, 261
called by intloopReadStreamConsole, 253
defun, 261
ncloopIncludeFileName, 861
called by ncloopInclude1, 861
calls concat, 861
calls incFileName, 861
defun, 861
ncloopInclude, 862
called by ncloopInclude1, 861
calls ncloopInclude0, 862
defun, 862
ncloopInclude0, 297
called by ncloopInclude, 862
calls incStream, 297
calls insertpile, 297
calls lineoftoks, 297
calls ncloopEchoParse, 297
calls ncloopProcess, 297
calls next, 297
uses $lines, 297
defun, 297
ncloopInclude1, 861
called by ncloopCommand, 750
calls ncloopIncludeFileName, 861
calls ncloopInclude, 861
defun, 861
ncloopParse, 262
called by parseFromString, 273
called by parseNoMacroFromString, 1531
calls dqToList, 262
calls ncloopDQlines, 262
calls npParse, 262
defun, 262
ncloopPrefix?, 750
called by ncloopCommand, 750
called by streamChop, 297
defun, 750
ncloopPrintLines, 294
called by intloopEchoParse, 294
defun, 294
ncloopProcess
called by ncloopInclude0, 297
ncParseFromString, 1240
called by conOpPage1, 1533
called by conSpecialString?, 1502
called by kPage, 1497
called by kcdePage, 1524
called by kcuPage, 1525
called by libConstructorSig, 1563
called by mkConform, 1532
defun, 1240
ncPutQ, 638
called by constoken, 337
called by intloopSpadProcess, 289
called by phIntReportMsgs, 291
called by phInterpret, 292
called by phMacro, 450
called by phParse, 291
called by setMsgCatlessAttr, 587
called by setMsgForcedAttr, 586
called by setMsgUnforcedAttr, 589
called by tokConstruct, 633
calls ncAlist, 639
calls ncTag, 639
calls qassq, 639
defun, 638
ncSoftError, 574
called by incHandleMessage, 301
called by mac0InfiniteExpansion, 453
called by macMacro, 458
called by npMissingMate, 440
called by npMissing, 376
called by npParse, 365
called by npTrapForm, 437
called by npTrap, 437
called by scanError, 358
called by scanS, 354
called by syIgnoredFromTo, 416
called by sySpecificErrorAtToken, 417
calls desiredMsg, 574
calls msgCreate, 574
calls processKeyedError, 574
uses $newcompErrorCount, 574
defun, 574
ncTag, 637
called by getMsgTag, 572
called by ncPutQ, 639
called by tokType, 635
<table>
<thead>
<tr>
<th>Function</th>
<th>Line</th>
<th>Called By</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>calls identp</td>
<td>637</td>
<td>ncTopLevel</td>
<td>248</td>
</tr>
<tr>
<td>calls ncBug</td>
<td>637</td>
<td>called by rnspad</td>
<td>243</td>
</tr>
<tr>
<td>calls qcar</td>
<td>637</td>
<td>calls ncIntLoop</td>
<td>248</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uses <em>eof</em>,</td>
<td>248</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uses $InteractiveFrame,</td>
<td>248</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uses $InteractiveMode,</td>
<td>248</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uses $boot,</td>
<td>248</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uses $e,</td>
<td>248</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uses $newspad</td>
<td>248</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uses $spad,</td>
<td>248</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uses in-stream</td>
<td>248</td>
</tr>
<tr>
<td></td>
<td></td>
<td>defun</td>
<td>248</td>
</tr>
<tr>
<td></td>
<td></td>
<td>new</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>newHelpSpad2Cmd</td>
<td>813</td>
</tr>
<tr>
<td></td>
<td></td>
<td>called by frameSpad2Cmd</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>called by abbreviatio?</td>
<td>813</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls concat</td>
<td>813</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls make-instream</td>
<td>813</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls makeInputFilename</td>
<td>813</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls namestring</td>
<td>813</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls obey</td>
<td>813</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls puame</td>
<td>813</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls poundsign</td>
<td>813</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls sayKeyedMsg</td>
<td>813</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls say</td>
<td>813</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls selectOptionLC</td>
<td>813</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls $syscommands</td>
<td>813</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls $useFullScreenHelp</td>
<td>813</td>
</tr>
<tr>
<td></td>
<td></td>
<td>defun</td>
<td>813</td>
</tr>
<tr>
<td></td>
<td></td>
<td>next</td>
<td>262</td>
</tr>
<tr>
<td></td>
<td></td>
<td>called by frameSpad2Cmd</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>called by intloopInclude0</td>
<td>288</td>
</tr>
<tr>
<td></td>
<td></td>
<td>called by intloopProcessString</td>
<td>261</td>
</tr>
<tr>
<td></td>
<td></td>
<td>called by ncloopInclude0</td>
<td>297</td>
</tr>
<tr>
<td></td>
<td></td>
<td>called by next1</td>
<td>263</td>
</tr>
<tr>
<td></td>
<td></td>
<td>called by parseFromString</td>
<td>273</td>
</tr>
<tr>
<td></td>
<td></td>
<td>called by parseNoMacroFromString</td>
<td>1531</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls Delay</td>
<td>262</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls next1</td>
<td>262</td>
</tr>
<tr>
<td></td>
<td></td>
<td>defun</td>
<td>262</td>
</tr>
<tr>
<td></td>
<td></td>
<td>next-lines-clear</td>
<td>1118</td>
</tr>
<tr>
<td></td>
<td></td>
<td>called by init-boot/spad-reader</td>
<td>1114</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uses boot-line-stack</td>
<td>1118</td>
</tr>
<tr>
<td></td>
<td></td>
<td>defun</td>
<td>1118</td>
</tr>
<tr>
<td></td>
<td></td>
<td>next-lines-show</td>
<td>1117</td>
</tr>
<tr>
<td></td>
<td></td>
<td>called by iostat</td>
<td>1116</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uses boot-line-stack</td>
<td>1117</td>
</tr>
<tr>
<td></td>
<td></td>
<td>defun</td>
<td>1117</td>
</tr>
<tr>
<td></td>
<td></td>
<td>next-token</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>used by token-stack-show</td>
<td>1117</td>
</tr>
<tr>
<td></td>
<td></td>
<td>next1</td>
<td>263</td>
</tr>
<tr>
<td></td>
<td></td>
<td>called by next</td>
<td>262</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls StreamNull</td>
<td>263</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls incAppend</td>
<td>263</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls next</td>
<td>263</td>
</tr>
<tr>
<td></td>
<td></td>
<td>defun</td>
<td>263</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nextInterpreterFrame</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>called by frameSpad2Cmd</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls updateFromCurrentInterpreterFrame</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uses $interpreterFrameRing</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>defun</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nextline</td>
<td>336</td>
</tr>
<tr>
<td></td>
<td></td>
<td>called by lineoftoks</td>
<td>335</td>
</tr>
<tr>
<td></td>
<td></td>
<td>called by scanEsc</td>
<td>347</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls npNull</td>
<td>336</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls strposl</td>
<td>336</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uses $f,</td>
<td>336</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uses $linepos</td>
<td>336</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uses $ln,</td>
<td>336</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uses $n,</td>
<td>336</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uses $r,</td>
<td>336</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uses $sz,</td>
<td>336</td>
</tr>
<tr>
<td></td>
<td></td>
<td>defun</td>
<td>336</td>
</tr>
</tbody>
</table>
nmsort
called by getWorkspaceNames, 727
nonBlank, 295
defun, 295
npADD, 382
called by npExpress1, 404
calls npAdd, 383
calls npPop1, 383
calls npPush, 383
calls npType, 382
defun, 382
npAdd, 383
called by npADD, 383
called by npPrimary2, 382
calls npCompMissing, 383
calls npDefinitionOrStatement, 383
calls npEqKey, 383
calls npEq Peek, 383
calls npPop1, 383
calls npPop2, 383
calls npPush, 383
calls npRestore, 383
calls npState, 383
calls npTrap, 383
calls npVariable, 383
calls pfAdd, 383
calls pfNothing, 383
defun, 383
npAmpersand, 428
called by npAmpersandFrom, 427
called by npAtom2, 384
calls npEqKey, 428
calls npName, 428
calls npTrap, 428
defun, 428
npAmpersandFrom, 427
called by npSynthetic, 423
calls npAmpersand, 427
calls npFromdom, 427
defun, 427
npAndOr, 406
called by npImport, 405
called by npInline, 398
called by npSuchThat, 401
called by npVoid, 403
called by npWhile, 402
calls npEqKey, 406
calls npPop1, 406
calls npPush, 406
calls npTrap, 406
defun, 406
npAngleBared, 411
called by npBracketed, 410
calls npEnclosed, 411
calls pfHide, 411
defun, 411
npAnyNo, 387
called by npColon, 442
called by npEncAp, 406
called by npTypified, 443
defun, 387
npApplication, 386
called by npFromdom1, 428
called by npFromdom, 427
called by npSCategory, 377
called by npTypedForm, 445
called by npTypified, 443
calls npApplication2, 386
calls npDotted, 386
calls npPop1, 386
calls npPop2, 386
calls npPrimary, 386
calls npPush, 386
calls pfApplication, 386
defun, 386
npApplication2, 387
called by npApplication2, 387
called by npApplication, 386
calls npApplication2, 387
calls npDotted, 387
calls npPop1, 388
calls npPop2, 388
calls npPrimary1, 387
calls npPush, 387
calls pfApplication, 388
defun, 387
npArith, 425
called by npInterval, 424
calls npLeftAssoc, 425
calls npSum, 425
defun, 425
npAssign, 411
called by npExit, 440
called by npPileExit, 440
calls npPileExit, 441
calls npBackTrack, 441
calls npMDEF, 441
defun, 441

npAssignment, 441
called by npAssign, 441
calls npAssignVariable, 441
calls npEqKey, 441
calls npGives, 441
calls npPop1, 441
calls npPop2, 441
calls npPush, 441
calls npTrap, 441
calls pfAssign, 441
defun, 441

npAssignVariable, 442
called by npAssign, 441
calls npColon, 442
calls npPop1, 442
calls npPop2, 442
calls npPush, 442
calls npTrap, 442
calls pfAssign, 441
defun, 442

npAtom1, 407
called by npPrimary1, 388
calls npBDefinition, 407
calls npConstTok, 407
calls npDollar, 407
calls npFromdom, 407
calls npName, 407
calls npPDefinition, 407
defun, 407

npAtom2, 384
called by npPrimary2, 382
calls npAmpersand, 384
calls npFromdom, 384
calls npInfixOperator, 384
calls npPrefixColon, 384
defun, 384

npBacksetElse, 421
called by npElse, 421
calls npEqKey, 421
defun, 421

npBackTrack, 372
called by npAssign, 441
called by npDefinitionOrStatement, 372
called by npExit, 440
called by npGives, 372
called by npMDEF, 389
calls npEqPeek, 372
calls npRestore, 372
calls npState, 372
calls npTrap, 372
defun, 372

npBDefinition, 409
called by npAtom1, 407
called by npEncl, 407
calls npBracketed, 409
calls npDefinitionlist, 409
calls npPDefinition, 409
defun, 409

npboot, 744
called by handleNoParseCommands, 742
defun, 744

npBPileDefinition, 412
called by npPrimary1, 388
calls npPileBracketed, 412
calls npPileDefinitionlist, 412
calls npPop1, 412
calls npPush, 412
calls pfListOf, 412
calls pfSequence, 412
defun, 412

npBraced, 411
called by npBracketed, 410
calls npEnclosed, 411
calls pfBraceBar, 411
calls pfBrace, 411
defun, 411

npBracked, 410
called by npBracketed, 410
calls npEnclosed, 410
calls pfBracketBar, 410
calls pfBracket, 410
defun, 410

npBracketed, 410
called by npBDefinition, 409
calls npAngleBared, 410
calls npBraced, 410
calls npBracked, 410
calls npParened, 410
defun, 410
npBreak, 399
called by npStatement, 395
calls npEqKey, 399
calls npPush, 399
calls pfBreak, 399
calls pfNothing, 399
defun, 399
npBy, 424
called by npForIn, 402
called by npSynthetic, 423
calls npInterval, 424
calls npLeftAssoc, 424
defun, 424
npCategory, 377
called by npCategoryL, 377
calls npPP, 377
calls npSCategory, 377
defun, 377
npCategoryL, 377
called by npSCategory, 377
called by npWith, 374
calls npCategory, 377
calls npPop1, 377
calls npPush, 377
calls pfUnSequence, 377
defun, 377
npCoercTo, 444
called by npTypeStyle, 443
calls npTypedForm, 444
calls pfCoerceto, 444
defun, 444
npColon, 442
called by npAssignVariable, 442
called by npPower, 427
calls npAnyNo, 442
calls npTagged, 442
calls npTypified, 442
defun, 442
npColonQuery, 444
called by npTypeStyle, 444
calls npTypedForm, 444
calls pfRetractTo, 444
defun, 444
npComma, 370
called by npQualDef, 369
calls npQualifiedDefinition, 370
calls npTuple, 370
defun, 370
npCommaBackSet, 370
called by npTuple, 370
calls npEqKey, 370
defun, 370
npCompMissing, 375
called by npAdd, 383
called by npForIn, 402
called by npLetQualified, 391
called by npLoop, 399
called by npWith, 375
calls npEqKey, 375
calls npMissing, 375
defun, 375
npConditional, 420
called by npConditionalStatement, 404
called by npWConditional, 419
calls npElse, 420
calls npEqKey, 420
calls npLogical, 420
calls npMissing, 420
calls npTrap, 420
defun, 420
npConditionalStatement, 404
called by npExpress1, 404
calls npConditional, 404
calls npQualifiedDefinition, 404
defun, 404
npConstTok, 408
called by npAtom1, 407
calls npEqPeek, 408
calls npNext, 408
calls npPop1, 409
calls npPrimary1, 409
calls npPush, 408
calls npRestore, 409
calls npState, 409
calls pfSymb, 409
calls tokPosn, 409
calls tokType, 408
uses $stok, 409
defun, 408
npDDInfKey, 432
called by npInfGeneric, 432
calls npEqKey, 432
calls npInfKey, 432
calls npPop1, 432
calls npPush, 432
calls npRestore, 432
calls npState, 432
calls pfSymb, 432
calls tokConstruct, 432
calls tokPart, 432
calls tokPosn, 432
uses $stok, 432
defun, 432
npDecl, 439
called by npVariableName, 438
calls npEqKey, 439
calls npPop1, 439
calls npPop2, 439
calls npPush, 439
calls npTrap, 439
calls npType, 439
calls pfTyped, 439
defun, 439
npDef, 412
called by npDefinitionItem, 392
called by npDefinitionOrStatement, 372
called by npDefn, 411
called by npFix, 390
calls npDefTail, 412
calls npMatch, 412
calls npPop1, 412
calls npPush, 412
calls npTrap, 412
calls pfCheckItOut, 412
calls pfDefinition, 412
calls pfPushBody, 412
defun, 412
npDefaultDecl, 394
called by npDefaultItem, 394
calls npEqKey, 394
calls npPop1, 394
calls npPop2, 394
calls npPush, 394
calls npTrap, 394
calls npType, 394
calls pfParts, 394
calls pfSpread, 394
defun, 394
npDefaultItem, 394
called by npSDefaultItem, 393
calls npDefaultDecl, 394
calls npTrap, 394
calls npTypeVariable, 394
defun, 394
npDefaultItemlist, 393
called by npTyping, 392
calls npPC, 393
calls npPop1, 393
calls npPush, 393
calls npSDefaultItem, 393
calls pfUnSequence, 393
defun, 393
npDefaultValue, 419
called by npSCategory, 377
calls npDefinitionOrStatement, 419
calls npEqKey, 419
calls npPop1, 419
calls npPush, 419
calls npTrap, 419
calls pfAdd, 419
calls pfNothing, 419
defun, 419
npDefinition, 391
called by npLetQualified, 391
called by npQualified, 371
calls npDefinitionItem, 391
calls npPP, 391
calls npPop1, 391
calls npPush, 391
calls pfSequenceToList, 391
defun, 391
npDefinitionItem, 392
called by npDefinition, 391
calls npDefn, 392
calls npDef, 392
calls npEqPeck, 392
calls npImport, 392
calls npMacro, 392
calls npRestore, 392
calls npState, 392
calls npTrap, 392
calls npTyping, 392
defun, 392

npDefinitionList, 417
called by npBDefinition, 409
called by npPDefinition, 408
called by npPileDefinitionList, 413
calls npQualDef, 417
calls npSemiListing, 417
defun, 417

npDefinitionOrStatement, 372
called by npAdd, 383
called by npDefTail, 419
called by npDefaultValue, 419
called by npLambda, 373
called by npLet, 391
called by npQualifiedDefinition, 371
calls npBackTrack, 372
calls npDef, 372
calls npGives, 372
defun, 372

npDefn, 411
called by npDefinitionItem, 392
called by npPrimary1, 388
calls npDef, 411
calls npEqKey, 411
calls npPP, 411
defun, 411

npDefTail, 419
called by npDef, 412
called by npMdef, 389
called by npSingleRule, 418
calls npDefinitionOrStatement, 419
calls npEqKey, 419
defun, 419

npDiscrim, 422
called by npDisjand, 422
calls npLeftAssoc, 422
calls npQuiver, 422
defun, 422

npDisjand, 422
called by npLogical, 421
calls npDiscrim, 422
calls npLeftAssoc, 422
defun, 422

npDollar, 408
called by npAtom1, 407
calls npEqPeek, 408
calls npNext, 408
calls npPush, 408
calls tokConstruct, 408
calls tokPosn, 408
uses $stok, 408
defun, 408

npDotted, 386
called by npApplication2, 387
called by npApplication, 386
defun, 386

npElse, 420
called by npConditional, 420
called by npBacksetElse, 421
calls npPop1, 421
calls npPop2, 421
calls npPop3, 421
calls npPush, 421
calls npRestore, 421
calls npState, 420
calls npTrap, 421
calls pfIfThenOnly, 421
calls pfif, 421
defun, 420

npEncAp, 406
called by npPrimary1, 388
called by npPrimary2, 382
calls npAnyNo, 406
calls npEncl, 406
calls npFromdom, 406
defun, 406

npEncl, 407
called by npEncAp, 406
calls npBDefinition, 407
calls npPop1, 407
calls npPop2, 407
calls npPush, 407
calls pfApplication, 407
defun, 407

npEnclosed, 436
called by npAngleBared, 411
called by npBraced, 411
called by npBracked, 410
called by npParened, 410
calls npEqKey, 436
calls npMissingMate, 436
calls npPop1, 436
calls npPush, 436
calls pfEnSequence, 436
calls pfListOf, 436
calls pfTuple, 436
uses $stok, 436
defun, 436
npEqKey, 369
called by npAdd, 383
called by npAndOr, 406
called by npAssignment, 441
called by npBacksetElse, 421
called by npBreak, 399
called by npCommaBackSet, 370
called by npCompMissing, 375
called by npConditional, 420
called by npDDInfixKey, 432
called by npDecl, 439
called by npDefTail, 419
called by npDefaultDecl, 394
called by npDefaultValue, 419
called by npDefn, 411
called by npEnclosed, 436
called by npExport, 395
called by npFix, 390
called by npForIn, 402
called by npFree, 398
called by npFromDom1, 428
called by npFromDom, 427
called by npInfGeneric, 432
called by npInfixOperator, 384
called by npItem1, 366
called by npItem, 366
called by npIterate, 399
called by npLambda, 373
called by npLetQualified, 391
called by npListAndRecover, 414
called by npList, 379
called by npLocalDecl, 397
called by npLocal, 397
called by npLoop, 399
called by npMoveTo, 415
called by npParentesize, 439
called by npPileBracketed, 413
called by npPileExit, 441
called by npQualified, 371
called by npReturn, 402, 403
called by npRule, 418
called by npSelector, 387
called by npSemiBackSet, 418
called by npSigDecl, 381
called by npSymbolVariable, 430
called by npTypedFrom1, 443
called by npTypedForm, 445
called by npTyping, 392
called by npWith, 374
calls npNext, 369
uses $stok, 369
uses $ttok, 369
defun, 369
npEqPeek, 376

called by npAdd, 383
called by npBackTrack, 372
called by npConstTok, 408
called by npDefinitionItem, 392
called by npDollar, 408
called by npInterval, 424
called by npListAndRecover, 414
called by npMoveTo, 415
called by npPrefixColon, 385
called by npSCategory, 377
called by npSegment, 425
called by npWith, 374
uses $stok, 376
uses $ttok, 376
defun, 376

npExit, 440
called by npGives, 372
calls npAssign, 440
calls npBackTrack, 440
calls npPileExit, 440
defun, 440
npExport, 395
called by npStatement, 395
called by npEqKey, 395
called by npLocalItemlist, 395
called by npPop1, 395
called by npPush, 395
called by npTrap, 395
called by pfExport, 395
defun, 395
Index

npExpress, 403
called by npReturn, 403
called by npStatement, 395
calls npExpress1, 403
calls npIterators, 404
calls npPop1, 404
calls npPop2, 404
calls npPush, 404
calls pfCollect, 404
calls pfListOf, 404
defun, 403

npExpress1, 404
called by npExpress, 403
calls npADD, 404
calls npConditionalStatement, 404
defun, 404

npFirstTok, 367
called by npNext, 370
called by npParse, 365
called by npRecoverTrap, 415
called by npRestore, 376
calls tokConstruct, 367
calls tokPart, 367
calls tokPosn, 367
uses $inputStream, 367
uses $stok, 367
uses $ttok, 367
defun, 367

npFix, 390
called by npPrimary1, 388
calls npDef, 390
calls npEqKey, 390
calls npPop1, 390
calls npPush, 390
calls pfFix, 390
defun, 390

npForIn, 402
called by npIterators, 400
called by npIterator, 400
calls npBy, 402
calls npCompMissing, 402
calls npEqKey, 402
calls npPop1, 402
calls npPop2, 402
calls npPush, 402
calls npTrap, 402
calls npVariable, 402
calls pfForIn, 402
defun, 402

npFree, 398
called by npStatement, 395
calls npEqKey, 398
calls npLocalItemlist, 398
calls npPop1, 398
calls npPush, 398
calls npTrap, 398
calls pfFree, 398
defun, 398

npFromdom, 427
called by npAmpersandFrom, 427
called by npAtom1, 407
called by npAtom2, 384
called by npEncAp, 406
called by npSegment, 425
calls npApplication, 427
calls npEqKey, 427
calls npFromdom1, 427
calls npPop1, 427
calls npPush, 427
calls npTrap, 427
calls pfFromDom, 427
defun, 427

npFromdom1, 428
called by npFromdom1, 428
called by npFromdom, 427
calls npApplication, 428
calls npEqKey, 428
calls npFromdom1, 428
calls npPop1, 428
calls npPush, 428
calls npTrap, 428
calls pfFromDom, 428
defun, 428

npGives, 372
called by npAssignment, 441
called by npDefinitionOrStatement, 372
calls npBackTrack, 372
calls npExit, 372
calls npLambda, 372
defun, 372

npId, 429
called by npName, 428
called by npSymbolVariable, 430
calls npNext, 429
calls npPush, 429
calls tokConstruct, 429
calls tokPosn, 429
uses $npTokToNames, 429
uses $stok, 429
uses $ttok, 429
defun, 429
npImport, 405
called by npDefinitionItem, 392
called by npStatement, 395
calls npAndOr, 405
calls npQualTypelist, 405
calls pfImport, 405
defun, 405
npInfGeneric, 432
called by npLeftAssoc, 431
called by npRightAssoc, 430
called by npTerm, 425
calls npDDInfKey, 432
calls npEqKey, 432
defun, 432
npInffixOp, 385
called by npInffixOperator, 384
calls npPushId, 385
uses $stok, 385
uses $ttok, 385
defun, 385
npInffixOperator, 384
called by npAtom2, 384
called by npSignatureDefinee, 381
calls npEqKey, 384
calls npInffixOp, 384
calls npPop1, 384
calls npPush, 384
calls npRestore, 384
calls npState, 384
calls pfSymb, 384
calls tokConstruct, 384
calls tokPart, 384
calls tokPosn, 384
uses $stok, 384
defun, 384
npInfKey, 433
called by npDDInfKey, 432
calls npPushId, 433
uses $stok, 433
uses $ttok, 433
defun, 433
npInline, 398
called by npStatement, 395
calls npAndOr, 398
calls npQualTypelist, 398
calls pfInline, 398
defun, 398
npInterval, 424
called by npBy, 424
calls npArith, 424
calls npEqPeek, 424
calls npPop1, 424
calls npPop2, 424
calls npPush, 424
calls npSegment, 424
calls pfApplication, 424
calls pfInfApplication, 424
defun, 424
npItem, 366
called by npParse, 365
calls npEqKey, 366
calls npItem1, 366
calls npPop1, 366
calls npPush, 366
calls npQualDef, 366
calls pfEnSequence, 366
calls pfNoValue, 366
defun, 366
npItem1, 366
called by npItem1, 367
called by npItem, 366
calls npEqKey, 366
calls npItem1, 367
calls npPop1, 367
calls npQualDef, 366
defun, 366
npIterate, 399
called by npStatement, 395
calls npEqKey, 399
calls npPush, 399
calls pfIterate, 399
calls pfNothing, 399
defun, 399
calls npPush, 414
calls npRecoverTrap, 414
calls syGeneralErrorHere, 414
uses $inputStream, 414
uses $stack, 414
catches, 414
defun, 414
npListing, 379
calledby npSDefaultItem, 393
calledby npSLocalItem, 396
calledby npSQualTypelist, 405
calledby npSigItemlist, 378
calledby npTypeVariablelist, 381
calledby npVariablelist, 438
calls npList, 379
calls pfListOf, 379
defun, 379
npListofFun, 445
calledby npSemiListing, 417
calledby npTuple, 370
calls npPop1, 445
calls npPop2, 445
calls npPop3, 445
calls npPush, 445
calls npTrap, 445
uses $stack, 445
defun, 445
npLocal, 397
calledby npStatement, 395
calls npEqKey, 397
calls npLocalItemlist, 397
calls npPop1, 398
calls npPush, 398
calls npTrap, 397
calls pLocal, 398
defun, 397
npLocalDecl, 397
calledby npSLocalItem, 396
calls npLocalDecl, 397
calls npTypeVariable, 397
defun, 397
npLocalItem, 397
calledby npSLocalItem, 396
calledby npLocalDecl, 397
calls npTypeVariable, 397
defun, 397
npLocalItemlist, 396
calledby npExport, 395
calledby npFree, 398
calledby npLocal, 397
calls npPC, 396
calls npPop1, 396
calls npPush, 396
calls npSLocalItem, 396
calls pfUnSequence, 396
defun, 396
npLogical, 421
calledby npConditional, 420
calledby npSuchThat, 401
calledby npSuch, 374
calledby npWhile, 402
calls npDisjand, 421
calls npLeftAssoc, 421
defun, 421
npLoop, 399
calledby npStatement, 395
calls npAssign, 399
calls npCompMissing, 399
calls npEqKey, 399
calls npIterators, 399
calls npPop1, 399
calls npPop2, 399
calls npPush, 399
calls npTrap, 399
calls pLoop1, 399
calls pLP, 399
defun, 399
npMacro, 388
calledby npDefinitionItem, 392
calledby npPrimary1, 388
calls npDef, 389
calls npMdef, 389
calls npPP, 388
defun, 388
npMatch, 374
calledby npDef, 412
called by npType, 373
calls npLeftAssoc, 374
calls npSuch, 374
defun, 374

npMDEF, 389
called by npAssign, 441
calls npBackTrack, 389
calls npMDEFinition, 390
calls npStatement, 390
defun, 389

npMdef, 389
called by npMDEFinition, 390
called by npMacro, 389
calls npDefTail, 389
calls npPop1, 389
calls npPush, 389
calls npQuiver, 389
calls npTrap, 389
calls pfCheckMacroOut, 389
calls pfMacro, 389
calls pfPushMacroBody, 389
defun, 389

npMDEFinition, 390
called by npMDEF, 390
calls npMdef, 390
calls npPP, 390
defun, 390

npMissing, 375
called by npCompMissing, 375
called by npConditional, 420
called by npMissingMate, 440
called by npPileBracketed, 413
calls ncSoftError, 376
calls pname, 376
calls tokPosn, 376
uses $stok, 376
defun, 375
throws, 375

npMissingMate, 440
called by npEnclosed, 436
called by npParenthesized, 439
calls ncSoftError, 440
calls npMissing, 440
calls tokPosn, 440
defun, 440

npMoveTo, 415
called by npMoveTo, 415
called by npRecoverTrap, 415
calls npEqKey, 415
calls npEqPeek, 415
calls npMoveTo, 415
calls npNext, 415
uses $inputStream, 415
defun, 415

npName, 428
called by npAmpersand, 428
called by npAtom1, 407
called by npReturn, 403
called by npSignatureDefinee, 381
called by npVariableName, 438
calls npId, 428
calls npSymbolVariable, 429
defun, 428

npNext, 369
called by npConstTok, 408
called by npDollar, 408
called by npEqKey, 369
called by npId, 429
called by npListAndRecover, 414
called by npMoveTo, 415
called by npPrefixColon, 386
called by npPushId, 433
calls npFirstTok, 370
uses $inputStream, 370
defun, 369

npNull, 555
called by eqpileTree, 561
called by insertpile, 557
called by nextline, 336
called by pileForests, 560
called by pilePlusComments, 558
called by pileTree, 559
calls StreamNull, 555
defun, 555

npParened, 410
called by npBracketed, 410
called by npPP, 434
calls npEnclosed, 410
calls pfParen, 410
defun, 410

npParenthesized, 439
called by npParenthesized, 439
calls npEqKey, 439
calls npMissingMate, 439
calls npPush, 439
uses $stok, 439
defun, 439
npParenthesized, 439
called by npPDefinition, 408
called by npTypeVariable, 380
called by npVariable, 437
calls npParenthesize, 439
defun, 439
npParse, 365
called by intloopEchoParse, 294
called by ncloopParse, 262
calls ncSoftError, 365
calls npFirstTok, 365
calls npItem, 365
calls pfDocument, 365
calls pListOf, 365
calls pfWrong, 365
calls tokPosn, 365
uses $inputStream, 365
uses $stack, 365
uses $stok, 365
uses $ttok, 365
catches, 365
defun, 365
npPC
called by npDefaultItemlist, 393
called by npLocalItemlist, 396
called by npQualTypelist, 405
npPDefinition, 408
called by npAtom1, 407
called by npBDefinition, 409
calls npDefinitionlist, 408
calls npParenthesized, 408
calls npPop1, 408
calls npPush, 408
calls pfEnSequence, 408
defun, 408
npPileBracketed, 413
called by npBPileDefinition, 412
called by npPP, 434
calls npEqKey, 413
calls npMissing, 413
calls npPop1, 413
calls npPush, 413
calls pfNothing, 413
calls pfPile, 413
defun, 413
npPileDefinitionlist, 413
called by npBPileDefinition, 412
calls npDefinitionlist, 413
calls npListAndRecover, 413
calls npPop1, 413
calls npPush, 413
calls pfAppend, 413
defun, 413
npPileExit, 440
called by npExit, 440
calls npAssign, 440
calls npEqKey, 441
calls npPop1, 441
calls npPop2, 441
calls npPush, 441
calls npStatement, 441
calls pfExit, 441
defun, 440
npPop1, 368
called by npADD, 383
called by npAdd, 383
called by npAndOr, 406
called by npApplication2, 388
called by npApplication, 386
called by npAssignVariable, 442
called by npAssignment, 441
called by npBPileDefinition, 412
called by npCategoryL, 377
called by npConstTok, 409
called by npDDInfKey, 432
called by npDecl, 439
called by npDefaultDecl, 394
called by npDefaultItemlist, 393
called by npDefaultValue, 419
called by npDefinition, 391
called by npDef, 412
called by npElse, 421
called by npEnclosed, 436
called by npEncl, 407
called by npExport, 395
called by npExpress, 404
called by npFix, 390
called by npForIn, 402
called by npFree, 398
called by npFromdom1, 428
called by npFromdom, 427
called by npInfixOperator, 384
called by npInterval, 424
called by npItem1, 367
called by npItem, 366
called by npIterators, 400
called by npLambda, 373
called by npLeftAssoc, 431
called by npLetQualified, 391
called by npListAndRecover, 414
called by npListofFun, 445
called by npList, 379
called by npLocalDecl, 397
called by npLocalItemList, 396
called by npLocal, 398
called by npLoop, 399
called by npMdef, 389
called by npPDefinition, 408
called by npPPff, 435
called by npPPLambda, 455
called by npPPLambda, 385
called by npPP, 434
called by npPileBracketed, 413
called by npPileDefinitionlist, 413
called by npPileExit, 441
called by npQualDef, 369
called by npQualTypelist, 405
called by npQualType, 406
called by npQualified, 371
called by npReturn, 403
called by npRightAssoc, 430
called by npSCategory, 377
called by npSDefaultItem, 393
called by npSLocalItem, 396
called by npSQualTypelist, 405
called by npSelector, 387
called by npSigDecl, 381
called by npSigItemList, 379
called by npSignature, 378
called by npSingleRule, 418
called by npSymbolVariable, 430
called by npSynthetic, 423
called by npTerm, 426
called by npTypeVariable, 380
called by npTypedForm1, 443
called by npTypedForm, 445
called by npType, 373
called by npTyping, 393
called by npVariableName, 438
called by npVariable, 438
called by npWConditional, 420
called by npWith, 375
called by npZeroOrMore, 401
uses $stack, 368
defun, 368

npPop2, 368
called by npAdd, 383
called by npApplication2, 388
called by npApplication, 386
called by npAssignment, 441
called by npDecl, 439
called by npDefaultDecl, 394
called by npElse, 421
called by npEncl, 407
called by npExpress, 404
called by npForIn, 402
called by npInterval, 424
called by npIterators, 400
called by npLambda, 373
called by npLeftAssoc, 431
called by npLetQualified, 391
called by npListofFun, 445
called by npList, 379
called by npLocalDecl, 397
called by npLoop, 399
called by npReturn, 403
called by npRightAssoc, 430
called by npSCategory, 377
called by npSDefaultItem, 393
called by npSLocalItem, 396
called by npSQualTypelist, 405
called by npSelector, 387
called by npSigDecl, 381
called by npSigItemList, 379
called by npSignature, 378
called by npSingleRule, 418
called by npSymbolVariable, 430
called by npSynthetic, 423
called by npTerm, 426
called by npTypedForm1, 443
called by npTypedForm, 445
called by npWith, 375
called by npZeroOrMore, 401
uses $stack, 368
defun, 368

npPop3, 368
INDEX

called by npElse, 421
called by npListofFun, 445
called by npList, 379
uses $stack, 368
defun, 368

npPower, 427
called by npProduct, 426
calls npColon, 427
calls npRightAssoc, 427
defun, 427

npPP, 434
called by npCategory, 377
called by npDefinition, 391
called by npDefn, 411
called by npMDEFinition, 390
called by npMacro, 388
called by npRule, 418
calls npPPf, 434
calls npPPg, 434
calls npParened, 434
calls npPileBracketed, 434
calls npPop1, 434
calls npPush, 434
calls pfEnSequence, 434
uses npFParg, 434
defun, 434

npPParg, 434
used by npPP, 434
defun, 434

npPPf, 435
called by npPPg, 435
called by npPP, 434
calls npPPf, 435
calls npSemiListing, 435
defun, 435

npPPff, 435
called by npPPf, 435
calls npPop1, 435
calls npPush, 435
uses $npPParg, 435
defun, 435

npPPg, 435
called by npPP, 434
calls npListAndRecover, 435
calls npPPi, 435
calls npPop1, 435
calls npPush, 435
uses $stack, 368
defun, 368
calls npList, 379
calls npRightAssoc, 427
defun, 427

npPrefixColon, 385
called by npAtom2, 384
called by npSignatureDefinee, 381
calls npEqPeek, 385
calls npNext, 386
calls npPush, 385
calls tokConstruct, 385
calls tokPosn, 386
uses $stok, 386
defun, 385

npPretend, 444
called by npTypeStyle, 444
calls npTypedForm, 444
calls pfPretend, 444
defun, 444

npPrimary, 382
called by npApplication, 386
called by npSCategory, 377
called by npSelector, 387
calls npPrimary1, 382
calls npPrimary2, 382
defun, 382

npPrimary1, 388
called by npApplication2, 387
called by npConstTok, 409
called by npPrimary, 382
calls npAtom1, 388
calls npBPileDefinition, 388
calls npDefn, 388
calls npEncAp, 388
calls npFix, 388
calls npLet, 388
calls npMacro, 388
calls npRule, 388
defun, 388

npPrimary2, 382
called by npPrimary, 382
calls npAdd, 382
calls npAtom2, 382
calls npEncAp, 382
calls npWith, 382
calls pfNothing, 382
defun, 382

npTyPeStyle, 444
calls npTypedForm, 444
calls pfPretend, 444
defun, 444

calls npEqPeek, 385
calls npNext, 386
calls npPush, 385
calls tokConstruct, 385
calls tokPosn, 386
uses $stok, 386
defun, 385

npTypeStyle, 444
calls npTypedForm, 444
calls pfPretend, 444
defun, 444

npTypedForm, 444
calls pfPretend, 444
defun, 444

npWith, 382
calls npPrimary2, 382
calls npPrimary, 382
calls npAdd, 382
calls npAtom2, 382
calls npEncAp, 382
calls npMacro, 388
calls npRule, 388
defun, 388
npProcessSynonym, 745
called by npSynonym, 745
calls printSynonyms, 745
calls processSynonymLine, 745
calls putalist, 745
calls terminateSystemCommand, 745
uses $CommandSynonymAlist, 745
defun, 745

npProduct, 426
called by npRemainder, 426
calls npLeftAssoc, 426
calls npPower, 426
defun, 426

cmpush, 367
called by npADD, 383
called by npAdd, 383
called by npAndOr, 406
called by npApplication2, 387
called by npApplication, 386
called by npAssignVariable, 442
called by npAssignment, 411
called by npBDefinition, 412
called by npBreak, 399
called by npCategoryL, 377
called by npConstTok, 408
called by npDDInfKey, 432
called by npDecl, 439
called by npDefaultDecl, 394
called by npDefaultItemlist, 393
called by npDefaultValue, 419
called by npDefinition, 391
called by npDef, 412
called by npDollar, 408
called by npElse, 421
called by npEnclosed, 436
called by npEncl, 407
called by npExport, 395
called by npExpress, 404
called by npFix, 390
called by npForIn, 402
called by npFree, 398
called by npFromdomain1, 428
called by npFromdomain, 427
called by npId, 429
called by npInfixOperator, 384
called by npInterval, 424
called by npItem, 366
called by npIterate, 399
called by npIterators, 400
called by npLambda, 373
called by npLeftAssoc, 431
called by npLetQualified, 391
called by npListAndRecover, 414
called by npListofFun, 445
called by npList, 379
called by npLocalDecl, 397
called by npLocalItemlist, 396
called by npLocal, 398
called by npLoop, 399
called by npMdef, 389
called by npPDefinition, 408
called by npPDef, 435
called by npPDef, 435
called by npPP, 434
called by npParenthesize, 399
called by npPileBracketed, 413
called by npPileDefinitionlist, 413
called by npPileExit, 441
called by npPrefixColon, 385
called by npQualDef, 369
called by npQualTypelist, 405
called by npQualType, 406
called by npQualified, 371
called by npRecoverTrap, 415
called by npReturn, 403
called by npRightAssoc, 430
called by npSCategory, 377
called by npSDefaultItem, 393
called by npSDefaultItem, 393
called by npSLocalItem, 396
called by npSLocalItem, 396
called by npSelector, 387
called by npSigDecl, 381
called by npSigItemlist, 378
called by npSignature, 378
called by npSingleRule, 418
called by npSymbolVariable, 430
called by npSynthetic, 423
called by npTerm, 426
called by npTypeVariable, 380
called by npTypedForm1, 443
called by npTypedForm, 445
called by npType, 373
INDEX

 called by npTyping, 393
called by npVariableName, 438
called by npVariable, 438
called by npWConditional, 420
called by npWith, 375
called by npZeroOrMore, 401
uses $stack, 367
defun, 367
npPushId, 433
called by npInfKey, 433
called by npInfixOp, 385
called by npSegment, 425
calls npNext, 433
calls tokConstruct, 433
calls tokPosn, 433
uses $stack, 433
uses $stok, 433
uses $ttok, 433
defun, 433
npQualDef, 369
called by npDefinitionlist, 417
called by npItem1, 366
called by npItem, 366
calls npComma, 369
calls npPop1, 369
calls npPush, 369
defun, 369
npQualified, 371
called by npQualifiedDefinition, 371
calls npDefinition, 371
calls npEqKey, 371
calls npLetQualified, 371
calls npPop1, 371
calls npPush, 371
calls npTrap, 371
calls pfWhere, 371
defun, 371
npQualifiedDefinition, 371
called by npComma, 370
called by npConditionalStatement, 404
calls npDefinitionOrStatement, 371
calls npQualified, 371
defun, 371
npQualType, 406
called by npSQualTypelist, 405
calls npPop1, 406
calls npPush, 406
calls npType, 406
calls pfNothing, 406
calls pfQualType, 406
defun, 406
npQualTypelist, 405
called by npImport, 405
called by npInline, 398
calls npPC, 405
calls npPop1, 405
calls npPush, 405
calls npSQualTypelist, 405
calls pfUnSequence, 405
defun, 405
npQuiver, 422
called by npDiscrim, 422
called by npMdef, 389
called by npSingleRule, 418
calls npRelation, 422
calls npRightAssoc, 422
defun, 422
npRecoverTrap, 415
called by npListAndRecover, 414
calls npFirstTok, 415
calls npMoveTo, 415
calls npPush, 415
calls pfDocument, 415
calls pfListOf, 415
calls pfWrong, 415
calls syIgnoredFromTo, 415
calls tokPosn, 415
uses $stok, 415
defun, 415
npRelation, 423
called by npQuiver, 422
calls npLeftAssoc, 423
calls npSynthetic, 423
defun, 423
npRemainder, 426
called by npTerm, 425
calls npLeftAssoc, 426
calls npProduct, 426
defun, 426
npRestore, 376
called by npAdd, 383
called by npBackTrack, 372
called by npSemiListing, 417
calls npEqKey, 418
defun, 418
npSemiListing, 417
called by npDefinitionlist, 417
called by npPPf, 435
calls npListofFun, 417
calls npSemiBackSet, 417
calls pfAppend, 417
defun, 417
npSigDecl, 381
called by npSigItem, 380
calls npEqKey, 381
calls npPop1, 381
calls npPop2, 381
calls npPush, 381
calls npTrap, 381
calls npType, 381
calls pfParts, 381
calls pfSpread, 381
defun, 381
npSigItem, 380
called by npSigItemList, 378
calls npSigDecl, 380
calls npTrap, 380
calls npTypeVariable, 380
defun, 380
npSigItemList, 378
called by npSignature, 378
calls npListing, 378
calls npPop1, 379
calls npPop2, 378
calls npSigItem, 378
calls pfAppend, 379
calls pfListof, 379
calls pfParts, 379
defun, 378
npSignature, 378
called by npSCategory, 377
calls npPop1, 378
calls npPush, 378
calls npSigItemList, 378
calls pfNothing, 378
calls pfWDec, 378
defun, 378
npSignatureDefinee, 381
called by npTypeVariableList, 381
called by npTypeVariable, 380
calls npInfixOperator, 381
calls npName, 381
calls npPrefixColon, 381
defun, 381
npSingleRule, 418
called by npRule, 418
calls npDefTail, 418
calls npPop1, 418
calls npPop2, 418
calls npPush, 418
calls npQuiver, 418
calls npTrap, 418
calls pfRule, 418
defun, 418
npSLocalItem, 396
called by npLocalItemList, 396
calls npListing, 396
calls npLocalItem, 396
calls npPop1, 396
calls npPop2, 396
calls npPush, 396
calls pfAppend, 396
calls pfParts, 396
defun, 396
npSQualTypelist, 405
called by npQualTypelist, 405
calls npListing, 405
calls npPop1, 405
calls npPop2, 405
calls npPush, 405
calls npQuiver, 405
calls pfParts, 405
defun, 405
npState, 436
called by npAdd, 383
called by npBackTrack, 372
called by npConstTok, 409
called by npDDInfKey, 432
called by npDefinitionItem, 392
called by npElse, 420
called by npInfixOperator, 384
called by npRightAssoc, 430
called by npSCategory, 377
called by npSymbolVariable, 430
called by npWith, 374
uses $inputStream, 436
uses $stack, 436
defun, 436

npStatement, 395
called by npDefinitionItem, 392
called by npMDEF, 390
called by npPileExit, 441
called by npVoid, 403
calls npBreak, 395
calls npExport, 395
calls npExpress, 395
calls npImport, 395
calls npInline, 395
calls npIterate, 395
calls npLocal, 395
calls npLoop, 395
calls npReturn, 395
calls npTyping, 395
calls npVoid, 395
defun, 395
	npSuch, 374
called by npMatch, 374
calls npLeftAssoc, 374
calls npLogical, 374
defun, 374
	npSuchThat, 401
called by npIterator, 400
calls npAndOr, 401
calls npLogical, 401
calls pfSuchthat, 401
defun, 401
	npSum, 425
called by npArith, 425
calls npLeftAssoc, 425
calls npTerm, 425
defun, 425
	npSymbolVariable, 430
called by npName, 429
calls npEqKey, 430
calls npId, 430
calls npPop1, 430
calls npPush, 430
calls npRestore, 430
calls npState, 430
calls tokConstruct, 430
calls tokPart, 430
calls tokPosn, 430
defun, 430
	npsynonym, 745
called by handleNoParseCommands, 742
calls npProcessSynonym, 745
defun, 745

npSynthetic, 423
called by npRelation, 423
calls npAmpersandFrom, 423
calls npBy, 423
calls npPop1, 423
calls npPop2, 423
calls npPush, 423
calls pfApplication, 423
calls pfInfApplication, 423
defun, 423

npsystem, 744
called by handleNoParseCommands, 742
calls sayKeyedMsg, 744
defun, 744

npTagged, 442
called by npColon, 442
calls npTypedForm1, 442
calls pfTagged, 442
defun, 442

npTerm, 425
called by npSum, 425
calls npInfGeneric, 425
calls npPop1, 426
calls npPop2, 426
calls npPush, 426
calls npRemainder, 425
calls pfApplication, 426
defun, 425

npTrap, 437
called by npAdd, 383
called by npAmpersand, 428
called by npAndOr, 406
called by npAssignment, 441
called by npBackTrack, 372
called by npConditional, 420
called by npDecl, 439
called by npDefaultValue, 419
called by npDefaultDecl, 394
called by npDefaultItem, 392
called by npDef, 412
called by npElse, 421
called by npExport, 395
called by npForIn, 402
called by npFree, 398
called by npFromDom1, 428
called by npFromDom, 427
called by npLambda, 373
called by npLetQualified, 391
called by npListofFun, 445
called by npList, 379
called by npLocalDecl, 397
called by npLocal, 397
called by npLoop, 399
called by npMdef, 389
called by npQualified, 371
called by npReturn, 403
called by npSCategory, 378
called by npSelector, 387
called by npSigDecl, 381
called by npSigItem, 380
called by npSingleRule, 418
called by npTypedForm1, 443
called by npTypedForm, 445
called by npTyping, 392
called by npWith, 374
calls ncSoftError, 437
calls tokPosn, 437
uses $stok, 437
defun, 437
throws, 437
npTrapForm, 437
called by pfCheckId, 471
called by pfCheckInOut, 469
called by pfCheckMacroOut, 470
calls ncSoftError, 437
calls pfSourceStok, 437
calls syGeneralErrorHere, 437
calls tokPosn, 437
defun, 437
throws, 437
npTuple, 370
called by npComma, 370
calls npCommaBackSet, 370
calls npListofFun, 370
calls pfTupleListOf, 370
defun, 370

npType, 373
called by npADD, 382
called by npDecl, 439
called by npDefaultDecl, 394
called by npLambda, 373
called by npLocalDecl, 397
called by npQualType, 406
called by npSigDecl, 381
called by npTypedForm1, 443
calls npMatch, 373
calls npPop1, 373
calls npPush, 373
calls npWith, 373
defun, 373

npTypedForm, 445
called by npCoerceTo, 444
called by npColonQuery, 444
called by npPretend, 444
called by npRestrict, 445
calls npApplication, 445
calls npEqKey, 445
calls npPop1, 445
calls npPop2, 445
calls npPush, 445
calls npTrap, 445
defun, 445

npTypedForm1, 443
called by npTagged, 442
calls npEqKey, 443
calls npPop1, 443
calls npPop2, 443
calls npPush, 443
calls npTrap, 443
calls npType, 443
defun, 443

npTypeStyle, 443
called by npTypified, 443
calls npCoerceTo, 443
calls npColonQuery, 444
calls npPretend, 444
calls npRestrict, 443
defun, 443

npTypeVariable, 380
called by npDefaultItem, 394
called by npLocalItem, 397
INDEX

called by npSigItem, 380
  calls npParenthesized, 380
calls npPop1, 380
  calls npPush, 380
calls npSignatureDefine, 380
calls npTypeVariableList, 380
calls pfListOf, 380
defun, 380

npTypeVariableList, 381
  called by npTypeVariable, 380
calls npListing, 381
calls npSignatureDefine, 381
defun, 381

npTypified, 443
  called by npColon, 442
calls npAnyNo, 443
calls npApplication, 443
calls npTypeStyle, 443
defun, 443

npTyping, 392
  called by npDefinitionItem, 392
called by npStatement, 395
calls npDefaultItemlist, 392
calls npEqKey, 392
calls npPop1, 393
calls npPush, 393
calls npTrap, 392
calls pfTyping, 393
defun, 392

npVariable, 437
  called by npAdd, 383
called by npForIn, 402
called by npLambda, 373
called by npWith, 375
calls npParenthesized, 437
calls npPop1, 438
calls npPush, 438
calls npVariableName, 438
calls npVariableList, 437
calls pfListOf, 438
defun, 437

npVariableList, 438
  called by npVariable, 437
calls npListing, 438
calls npVariableName, 438
defun, 438

npVariableName, 438
  called by npVariableList, 438
called by npVariable, 438
calls npDecl, 438
calls npName, 438
calls npPop1, 438
calls npPush, 438
calls pfNothing, 438
calls pfTyped, 438
defun, 438

npVoid, 403
  called by npStatement, 395
calls npAndOr, 403
calls npStatement, 403
calls pfNoValue, 403
defun, 403

npWConditional, 419
  called by npSCategory, 377
calls npConditional, 419
calls npPop1, 420
calls npPush, 420
calls pfTweakIf, 420
defun, 419

npWhile, 402
  called by npIterators, 400
called by npIterator, 401
calls npAndOr, 402
calls npLogical, 402
calls pfWhile, 402
defun, 402

npWith, 374
  called by npPrimary2, 382
called by npType, 373
calls npCategoryL, 374
calls npCompMissing, 375
calls npEqKey, 374
calls npEqPeek, 374
calls npPop1, 375
calls npPop2, 375
calls npPush, 375
calls npRestore, 375
calls npState, 374
calls npTrap, 374
calls npVariable, 375
calls pfNothing, 375
calls pfWith, 375
INDEX

defun, 374
npZeroOrMore, 401
called by npIterators, 400
calls npPop1, 401
calls npPop2, 401
calls npPush, 401
uses $stack, 401
defun, 401

nreverse0
called by augmentHasArgs, 1523
called by bcInputExplicitMatrix, 1339
called by bcMakeEquations, 1388
called by bcMakeLinearEquations, 1388
called by coerceInt1, 676
called by reduceAlistForDomain, 1520
called by reportOpsFromLisplib, 1024
called by reportOpsFromUnitDirectly, 1028

NRTcompileEvalForm
called by coerceInt1, 675

NRTevalDomain, 1189
called by compiledLookup, 1185
calls evalDomain, 1189
calls eval, 1189
calls qcar, 1189
defun, 1189

nsubst
called by ruleLhsTran, 552
called by zeroOneTran, 292

obey
called by copyright, 787
called by editFile, 805
called by license, 866
called by newHelpSpad2Cmd, 813
called by summary, 1035
objCodeMode, 449
defmacro, 449

objCodeVal, 449
defmacro, 449

object2Identifier
called by fixObjectForPrinting, 728
called by frameSpad2Cmd, 11
called by pathnameTypeId, 1191
called by readHiFi, 843
called by saveHistory, 837
called by selectOptionLC, 751
called by setHistoryCore, 826
called by writeHiFi, 844

object2String
called by clearCmdExcept, 772
called by displayMacro, 726
called by displaySetOptionInformation, 896
called by displaySetVariableSettings, 897
called by getTraceOption, 55
called by makePathname, 1193
called by set1, 1015
called by setExposeAddGroup, 89
called by setFunctionsCache, 911
called by setLinkerArgs, 933
called by setNagHost, 959
called by setOutputAlgebra, 967
called by setOutputFormula, 995
called by setOutputFortran, 974
called by setOutputHtml, 980
called by setOutputMathml, 986
called by setOutputOpenMath, 991
called by setOutputTex, 1002
called by setStreamsCalculate, 1007

objEnv, 449
defun, 449

objMode, 448
defun, 449
defmacro, 449

called by coerceBranch2Union, 698
called by coerceInt0, 674
called by coerceInt1, 676
called by coerceIntAlgebraicConstant, 698
called by coerceIntByMap, 693
called by coerceIntCommutate, 689
called by coerceIntPermute, 685
called by coerceIntSpecial, 692
called by coerceIntTableOrFunction, 690
called by coerceInteractive, 672
called by coerceInt, 673
called by coerceRetract, 709
called by coerceUnion2Branch, 707
called by displayType, 732
called by displayValue, 731
called by getAndEvalConstructorArgument, 1104
called by interpret2, 280
called by isDomainValuedVariable, 1105
called by printTypeAndTime, 284
called by processInteractive1, 277
called by retract2Specialization, 704  
called by retractByFunction, 710  
called by retract, 1203  
called by showInOut, 842  
called by transTraceItem, 60  
defmacro, 448  
objModeFn, 1238  
defun, 1238  
objSetMode, 448  
called by coerceInt0, 674  
defmacro, 448  
objSetVal, 447  
defmacro, 447  
objVal, 448  
called by coerceBranch2Union, 698  
called by coerceInt0, 674  
called by coerceInt1, 676  
called by coerceIntByMap, 693  
called by coerceIntTableOrFunction, 691 oldCompLookup, 1189  
called by coerceIntInteractive, 672  
called by coerceInt, 673  
called by coerceUnion2Branch, 707  
called by evaluateType, 1078  
called by getAndEvalConstructorArgument, 819 oldHistFileName, 819  
called by evaluateType1, 1080  
called by isDomainValuedVariable, 1105  
called by processInteractive1, 277  
called by retract2Specialization, 704  
called by retractByFunction, 710  
called by showInOut, 842  
called by valueArgsEqual?, 696  
defmacro, 448 ofCategory, 649  
called by coerceIntAlgebraicConstant, 698  
called by evalCategory, 1105  
called by ofCategory, 649  
calls hasCaty, 649  
calls identp, 649  
called by coerceIntAlgebraicConstant, 698  
called by evalCategory, 1105  
called by ofCategory, 649  
called by basicLookup, 1186 local def $Subst, 649  
called by lookupInDomainVector, 1189  
called by $lookupDefaults, 1189  
defun, 1189  
called by initHist, 820  
called by makeHistFileName, 819 local def $hope, 649  
defun, 819  
called by oldHistFileName, 819 oldParseString, 1448  
defun, 1448  
called by initHist, 820  
called by oldHistFileName, 819 om-bindTCP, 1291 
defun, 1291  
called by oldHistFileName, 819 om-closeTCP, 1291 
defun, 1291  
called by oldHistFileName, 819 om-closeConn, 1290 
defun, 1290  
called by oldHistFileName, 819 om-closeDev, 1290  
defun, 1290  
called by oldHistFileName, 819 om-connectTCP, 1292 
defun, 1292  
called by oldHistFileName, 819 om-getApp, 1293 
defun, 1293  
called by oldHistFileName, 819 om-getAtp, 1294 
defun, 1294  
called by oldHistFileName, 819 om-getAttr, 1294 
defun, 1294  
called by oldHistFileName, 819 om-getBind, 1294 
defun, 1294
om-getBVar, 1294
defun, 1294
om-getByteArray, 1295
defun, 1295
om-getConnInDev, 1291
defun, 1291
om-getConnOutDev, 1291
defun, 1291
om-getEndApp, 1295
defun, 1295
om-getEndAtp, 1295
defun, 1295
om-getEndAttr, 1295
defun, 1295
om-getEndBind, 1296
defun, 1296
om-getEndBVar, 1296
defun, 1296
om-getEndError, 1296
defun, 1296
om-getEndObject, 1296
defun, 1296
om-getError, 1297
defun, 1297
om-getFloat, 1297
defun, 1297
om-getInt, 1297
defun, 1297
om-getObject, 1297
defun, 1297
om-getString, 1298
defun, 1298
om-getSymbol, 1298
defun, 1298
om-getType, 1298
defun, 1298
om-getVar, 1298
defun, 1298
om-listCDs, 1288
defun, 1288
om-listSymbols, 1288
defun, 1288
om-makeConn, 1290
defun, 1290
om-openFileDev, 1289
defun, 1289
om-openStringDev, 1290
defun, 1290
om-putApp, 1299
defun, 1299
om-putAtp, 1299
defun, 1299
om-putAttr, 1299
defun, 1299
om-putBind, 1299
defun, 1299
om-putBVar, 1300
defun, 1300
om-putByteArray, 1300
defun, 1300
om-putEndApp, 1300
defun, 1300
om-putEndAtp, 1300
defun, 1300
om-putEndAttr, 1301
defun, 1301
om-putEndBind, 1301
defun, 1301
om-putEndBVar, 1301
defun, 1301
om-putEndError, 1301
defun, 1301
om-putEndObject, 1302
defun, 1302
om-putError, 1302
defun, 1302
om-putFloat, 1302
defun, 1302
om-putInt, 1302
defun, 1302
om-putObject, 1303
defun, 1303
om-putString, 1303
defun, 1303
om-putSymbol, 1303
defun, 1303
om-putVar, 1303
defun, 1303
om-Read, 1287
defun, 1287
om-setDevEncoding, 1289
defun, 1289
om-stringPtrToString, 1304
defun, 1304
om-stringToStringPtr, 1304
defun, 1304
om-supportsCD, 1288
defun, 1288
om-supportsSymbol, 1288
defun, 1288
openOutputLibrary, 905
calledby setOutputLibrary, 904
calls dropInputLibrary, 905
uses input-libraries, 905
uses output-library, 905
defun, 905
openserver, 1133
calledby restart, 239
defun, 1133
operationopen, 1153
calledby resethashtable, 1146
calledby restart0, 241
uses *operation-hash*, 1153
uses *operation-stream*, 1153
uses *operation-stream-stamp*, 1153
uses $spadroot, 1153
defun, 1153
opIsHasCat
calledby basicLookup, 1186
opOf
calledby abbreviationsSpad2Cmd, 755
calledby augmentHasArgs, 1523
calledby conOpPage1, 1533
calledby dbAddDocTable, 1540
calledby dbSearchOrder, 1514
calledby dbSelectCon, 1552
calledby dbShowConditions, 1552
calledby dbShowCons1, 1547
calledby dbShowConsDoc, 1549
calledby displaySpad2Cmdd, 796
calledby domainToGenvar, 1054
calledby getConstantFromDomain, 699
calledby hasCaty, 650
calledby isDomainOrPackage, 1061
calledby kArgumentCheck, 1529
calledby kPage, 1497
calledby kcPage, 1515
calledby kcaPage1, 1521
calledby kccPage, 1522
calledby kcdePage, 1524
calledby kcnPage, 1526
calledby kcpPage, 1519
calledby kcuPage, 1525
calledby kePage, 1509
calledby reportOperations, 1021
calledby spadTrace, 67
optionError, 721
calledby countCache, 912
calledby readSpad2Cmd, 875
calls commandErrorMessage, 721
defun, 721
optionUserLevelError, 722
calls userLevelErrorMessage, 722
defun, 722
opTran, 553
calledby pf2Sex1, 530
calledby pfApplication2Sex, 536
defun, 553
orderBySlotNumber, 1068
calls assocright, 1068
calls exit, 1068
calls orderList, 1068
calls seq, 1068
defun, 1068
orderList
calledby orderBySlotNumber, 1068
orderMms
calledby retractByFunction, 710
orderUnionEntries
calledby coerceBranch2Union, 698
calledby coerceUnion2Branch, 707
originsInOrder, 1540
calledby dbDocTable, 1539
calledby originsInOrder, 1540
calls ancestorsOf, 1540
calls assocleft, 1540
calls getdatabase, 1540
calls insert, 1540
calls originsInOrder, 1540
calls parentsOf, 1540
defun, 1540
out-stream, 1108
usedby spad-long-error, 1115
defvar, 1108
INDEX

output
calledby recordAndPrint, 282
calledby outputAndPrint, 282
output-library, 905
  usedby openOutputLibrary, 905
defvar, 905
outputFormat
calledby displayValue, 731

parallel, 621
syntax, 621
parentsOf
calledby kcpPage, 1519
calledby originsInOrder, 1540
parse-integer
calledby bcInputEquations, 1385
calledby bcInputExplicitMatrix, 1339
calledby bcInputMatrixByFormula, 1337
parseAndEval, 1447
defun, 1447
parseAndEval1, 1447
defun, 1447
parseAndInterpret, 272
calledby executeQuietCommand, 272
calledby serverReadLine, 269
  uses $InteractiveFrame, 272
  uses $InteractiveMode, 272
  uses $boot, 272
  uses $spad, 272
defun, 272
parseFromString, 273
calledby parseSystemCmd, 740
calls StreamNull, 273
calls incString, 273
calls lineoftoks, 273
calls macroExpanded, 273
calls nloopParse, 273
calls next, 273
calls pf2Sex, 273
defun, 273
parseNoMacroFromString, 1531
calledby mkConform, 1532
calls StreamNull, 1531
calls function, 1531
calls incString, 1531
calls lineoftoks, 1531
calls nloopParse, 1531
calls next, 1531
calls pf2Sex, 1531
defun, 1531
ParsePair, 142
parseSystemCmd, 740
calledby handleParsedSystemCommands, 740
calls dumbTokenize, 740
calls parseFromString, 740
calls stripSpaces, 740
calls tokTran, 740
defun, 740
parseWord, 1463
defun, 1463
pathname, 1192
calledby deleteFile, 1193
calledby editFile, 805
calledby editSpad2Cmd, 804
calledby getDependentsOfConstructor, 1525
calledby getUsersOfConstructor, 1526
calledby makePathname, 1193
calledby namestring, 1190
calledby pathnameDirectory, 1192
calledby pathnameName, 1191
calledby pathnameType, 1191
calledbyspathname, 1192
calledby readSpad2Cmd, 875
calledby reportOpsFromLispLib1, 1024
calledby reportOpsFromUnitDirectly1, 1032
calledby setExposeAddGroup, 89
calledby setExpose, 91
calledby updateSourceFiles, 806
calledby workfilesSpad2Cmd, 1099
calledby make-filename, 1192
calledby pathname, 1192
defun, 1192
pathnameDirectory, 1192
calledby editSpad2Cmd, 804
calledby loadLib, 1181
calledby mergePathnames, 1191
calledby setOutputAlgebra, 967
calledby setOutputFormula, 995
calledby setOutputFortran, 974
calledby setOutputHtml, 980
calledby setOutputMathml, 986
called by setOutputOpenMath, 991
called by setOutputTex, 1002
calls pathname, 1192
defun, 1192
pathnameName, 1191
called by editSpad2Cmd, 804
called by mergePathnames, 1191
called by readSpad2Cmd, 875
called by setOutputAlgebra, 967
called by setOutputFormula, 995
called by setOutputFortran, 974
called by setOutputHtml, 980
called by setOutputMathml, 986
called by setOutputOpenMath, 991
called by setOutputTex, 1002
called by updateSourceFiles, 806
calls pathname, 1191
defun, 1191
PathnameString
called by pfname, 316
pathnameType, 1191
called by editSpad2Cmd, 804
called by mergePathnames, 1191
called by pathnameTypeId, 1191
called by setOutputAlgebra, 967
called by setOutputFormula, 995
called by setOutputFortran, 974
called by setOutputHtml, 980
called by setOutputMathml, 986
called by setOutputOpenMath, 991
called by setOutputTex, 1002
called by updateSourceFiles, 806
calls pathname, 1191
defun, 1191
pathnameTypeId, 1191
called by readSpad2Cmd, 875
called by updateSourceFiles, 806
calls object2Identifier, 1191
calls pathnameType, 1191
calls upcase, 1191
defun, 1191
patternVarsOf, 551
called by ruleLhsTran, 552
called by rulePredicateTran, 550
calls patternVarsOf1, 551
defun, 551
patternVarsOf1, 551
called by patternVarsOf1, 551
called by patternVarsOf, 551
calls patternVarsOf1, 551
defun, 551
pcounters, 1053
called by trace1, 48
calls bright, 1053
calls concat, 1053
calls sayBrightly, 1053
uses /countlist, 1053
defun, 1053
peekTimedName
called by interpretTopLevel, 278
permuteToOrder, 687
called by computeTTTranspositions, 686
called by permuteToOrder, 687
calls permuteToOrder, 687
calls setelt, 687
defun, 687
pf0ApplicationArgs, 467
called by macApplication, 451
calls pf0FlattenSyntacticTuple, 467
calls pfApplicationArg, 467
defun, 467
pf0AssignLhsItems, 486
called by pf2Sex1, 531
calls pfAssignLhsItems, 486
calls pfParts, 486
defun, 486
pf0DefinitionLhsItems, 492
called by pfDefinition2Sex, 545
calls pfDefinitionLhsItems, 492
calls pfParts, 492
defun, 492
pf0FlattenSyntacticTuple, 467
called by pf0ApplicationArgs, 467
called by pf0FlattenSyntacticTuple, 467
calls pf0FlattenSyntacticTuple, 467
calls pf0TupleParts, 467
calls pfTuple?, 467
defun, 467
pf0ForinLhs, 497
called by pf2Sex1, 530
calls pfForinLhs, 497
calls pfParts, 497
INDEX

504
505
506
508
517
523
529
531
533
536
537
540
545
549
558
561
565
572
580
586
591
596
602
607
613
618
623
628
633
638
643
648
653
658
663
668
673
678
683
688
693
698
703
708
713
718
723
728
733
738
743
748
753
758
763
768
773
778
783
788
793
798
803
808
813
818
823
828
833
838
843
848
853
858
863
868
873
878
883
888
893
898
903
908
913
918
923
928
933
938
943
948
953
958
963
968
973
978
983
988
993
998
1003
1008
1013
1018
1023
1028
1033
1038
1043
1048
1053
1058
1063
1068
1073
1078
1083
1088
1093
1098
1103
1108
1113
1118
1123
1128
1133
1138
1143
1148
1153
1158
1163
1168
1173
1178
1183
1188
1193
1198
1203
1208
1213
1218
1223
1228
1233
1238
1243
1248
1253
1258
1263
1268
1273
1278
1283
1288
1293
1298
1303
1308
1313
1318
1323
1328
1333
1338
1343
1348
1353
1358
1363
1368
1373
1378
1383
1388
1393
1398
1403
1408
1413
1418
1423
1428
1433
1438
1443
1448
1453
1458
1463
1468
1473
1478
1483
1488
1493
1498
1503
1508
1513
1518
1523
1528
1533
1538
1543
1548
1553
1558
1563
1568
1573
1578
1583
1588
1593
1598
1603
1608
1613
1618
1623
1628
1633
1638
1643
1648
1653
1658
1663
1668
1673
1678
1683
1688
1693
1698
1703
1708
1713
1718
1723
1728
1733
1738
1743
1748
1753
1758
1763
1768
1773
1778
1783
<table>
<thead>
<tr>
<th>Function</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>pfAndLeft</td>
<td>531</td>
</tr>
<tr>
<td>pfAndRight</td>
<td>531</td>
</tr>
<tr>
<td>pfApplication2Sex</td>
<td>530</td>
</tr>
<tr>
<td>pfApplication?</td>
<td>530</td>
</tr>
<tr>
<td>pfAssign?</td>
<td>531</td>
</tr>
<tr>
<td>pfAssignRhs</td>
<td>531</td>
</tr>
<tr>
<td>pfBreak?</td>
<td>531</td>
</tr>
<tr>
<td>pfBreakFrom</td>
<td>531</td>
</tr>
<tr>
<td>pfCoerceTo?</td>
<td>530</td>
</tr>
<tr>
<td>pfCoerceToExpr</td>
<td>530</td>
</tr>
<tr>
<td>pfCoerceToType</td>
<td>530</td>
</tr>
<tr>
<td>pfCollect2Sex</td>
<td>530</td>
</tr>
<tr>
<td>pfCollect?</td>
<td>530</td>
</tr>
<tr>
<td>pfDefinition2Sex</td>
<td>531</td>
</tr>
<tr>
<td>pfDefinition?</td>
<td>531</td>
</tr>
<tr>
<td>pfDo?</td>
<td>531</td>
</tr>
<tr>
<td>pfDoBody</td>
<td>531</td>
</tr>
<tr>
<td>pfExit?</td>
<td>530</td>
</tr>
<tr>
<td>pfExitCond</td>
<td>530</td>
</tr>
<tr>
<td>pfExitExpr</td>
<td>530</td>
</tr>
<tr>
<td>pfForin?</td>
<td>530</td>
</tr>
<tr>
<td>pfForinWhole</td>
<td>530</td>
</tr>
<tr>
<td>pfFree?</td>
<td>531</td>
</tr>
<tr>
<td>pfFromdom?</td>
<td>530</td>
</tr>
<tr>
<td>pfFromdomDomain</td>
<td>530</td>
</tr>
<tr>
<td>pfFromdomWhat</td>
<td>530</td>
</tr>
<tr>
<td>pfIdSymbol</td>
<td>530</td>
</tr>
<tr>
<td>pfIf?</td>
<td>530</td>
</tr>
<tr>
<td>pfIfCond</td>
<td>530</td>
</tr>
<tr>
<td>pfIfElse</td>
<td>530</td>
</tr>
<tr>
<td>pfIfThen</td>
<td>530</td>
</tr>
<tr>
<td>pfIterate?</td>
<td>531</td>
</tr>
<tr>
<td>pfLambda2Sex</td>
<td>531</td>
</tr>
<tr>
<td>pfLambda?</td>
<td>531</td>
</tr>
<tr>
<td>pfLiteral2Sex</td>
<td>530</td>
</tr>
<tr>
<td>pfLiteral?</td>
<td>530</td>
</tr>
<tr>
<td>pfLocal?</td>
<td>531</td>
</tr>
<tr>
<td>pfLoop?</td>
<td>530</td>
</tr>
<tr>
<td>pfMLambda?</td>
<td>531</td>
</tr>
<tr>
<td>pfMacro?</td>
<td>531</td>
</tr>
<tr>
<td>pfNot?</td>
<td>531</td>
</tr>
<tr>
<td>pfNotArg</td>
<td>531</td>
</tr>
<tr>
<td>pfNothing?</td>
<td>530</td>
</tr>
<tr>
<td>pfNoValue?</td>
<td>531</td>
</tr>
<tr>
<td>pfNoValueExpr</td>
<td>531</td>
</tr>
<tr>
<td>pfOr?</td>
<td>531</td>
</tr>
</tbody>
</table>

Calls pfOrLeft, 531
Calls pfOrRight, 531
Calls pfPretend?, 530
Calls pfPretendExpr, 530
Calls pfPretendType, 530
Calls pfRestrict?, 531
Calls pfRestrictExpr, 531
Calls pfRestrictType, 531
Calls pfReturn?, 531
Calls pfReturnExpr, 531
Calls pfRule2Sex, 531
Calls pfRule?, 531
Calls pfSequence2Sex, 530
Calls pfSequence?, 530
Calls pfSuchthat?, 531
Calls pfSuchthatCond, 531
Calls pfSymbol?, 530
Calls pfSymbolSymbol, 530
Calls pfTagged?, 530
Calls pfTaggedExpr, 530
Calls pfTaggedTag, 530
Calls pfTuple?, 530
Calls pfTyped?, 531
Calls pfTypedId, 531
Calls pfTypedType, 531
Calls pfWhere?, 531
Calls pfWhereExpr, 531
Calls pfWhile?, 530
Calls pfWhileCond, 530
Calls pfWrong?, 531
Calls spadThrow, 531
Calls tokPart, 532
Uses $QuietCommand, 532
Uses $insideRule, 532
Uses $insideSEQ, 532
Defun, 530

pf2sex1 called by pfCollectArgTran, 547

pfAbSynOp, 634 called by macLambdaParameterHandling, 460
called by pf2Sex1, 532
called by pfCopyWithPos, 466
called by pfLeaf?, 477
called by pfLiteral?, 478
called by pfLiteralClass, 478
<table>
<thead>
<tr>
<th>Function</th>
<th>Page(s)</th>
<th>Called by</th>
</tr>
</thead>
<tbody>
<tr>
<td>pfAdd</td>
<td>482</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfAnd</td>
<td>483</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfApplication</td>
<td>483</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfAssign</td>
<td>486</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfBreak</td>
<td>488</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfCoerceTo</td>
<td>489</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfCollect</td>
<td>490</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfDefinition</td>
<td>491</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfDo</td>
<td>492</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfExit</td>
<td>493</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfForIn</td>
<td>496</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfFree</td>
<td>495</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfFromDom</td>
<td>497</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfId</td>
<td>476</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfIf</td>
<td>477</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfIterate</td>
<td>501</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfLambda</td>
<td>502</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfLocal</td>
<td>503</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfLoop</td>
<td>504</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfMLambda</td>
<td>505</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfMacro</td>
<td>506</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfNot</td>
<td>507</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfNothing</td>
<td>508</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfOr</td>
<td>509</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfPretend</td>
<td>510</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfRestrict</td>
<td>511</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfReturn</td>
<td>512</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfRule</td>
<td>513</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfSequence</td>
<td>514</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfSuchthat</td>
<td>515</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfSymbol</td>
<td>516</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfTagged</td>
<td>517</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfTuple</td>
<td>518</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfTyped</td>
<td>519</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfWhere</td>
<td>520</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfWhile</td>
<td>521</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfWrong</td>
<td>522</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>calls eqcar</td>
<td>634</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>defun</td>
<td>634</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>pfAbSynOp</td>
<td>634</td>
<td>pfMapParts, 467</td>
</tr>
<tr>
<td>calledby intloopProcess</td>
<td>288</td>
<td>pfMapParts, 467</td>
</tr>
</tbody>
</table>
calls pfTree, 483  
defun, 483  

pfApplication2Sex, 535  
called by pf2Sex1, 530  
calls hasOptArgs?, 536  
calls opTran, 536  
calls pf0TupleParts, 536  
calls pf2Sex1, 536  
calls pf2Sex, 536  
calls pfApplicationArg, 536  
calls pfApplicationOp, 536  
calls pfOp2Sex, 535  
calls pfSuchThat2Sex, 536  
calls ptTuple?, 536  
uses $insideApplication, 536  
uses $insideRule, 536  
defun, 535  

pfApplication?, 485  
called by macExpand, 451  
called by pf2Sex1, 530  
called by pfCheckItOut, 469  
called by pfCheckMacroOut, 470  
called by pfCollect1?, 472  
called by pfFlattenApp, 471  
called by pfFromDom, 497  
called by pfSexpr,strip, 480  
calls pfAbSynOp?, 485  
defun, 485  

pfApplicationArg, 484  
called by pf0ApplicationArgs, 467  
called by pfApplication2Sex, 536  
called by pfCollectVariable1, 472  
called by pfFlattenApp, 472  
called by pfFromDom, 497  
called by pfSexpr,strip, 480  
defun, 484  

pfApplicationOp, 484  
called by macApplication, 451  
called by pfApplication2Sex, 536  
called by pfCollect1?, 472  
called by pfFlattenApp, 472  
called by pfFromDom, 497  
called by pfSexpr,strip, 480  
defun, 484  

pfAssign, 485  
called by npAssignment, 441  
calls pfTree, 485  
defun, 485  

pfAssign?, 486  
called by pf2Sex1, 531  
calls pfAbSynOp?, 486  
defun, 486  

pfAssignLhsItems, 486  
called by pf0AssignLhsItems, 486  
defun, 486  

pfAssignRhs, 486  
called by pf2Sex1, 531  
defun, 486  

pfAttribute, 483  
called by npSCategory, 377  
calls pfTree, 483  
defun, 483  

pfBrace, 487  
called by npBraced, 411  
calls pfApplication, 487  
calls pfIdPos, 487  
calls tokPosn, 487  
defun, 487  

pfBraceBar, 487  
called by npBraced, 411  
calls pfApplication, 487  
calls pfIdPos, 487  
calls tokPosn, 487  
defun, 487  

pfBracket, 487  
called by npBracked, 410  
calls pfApplication, 487  
calls pfIdPos, 487  
calls tokPosn, 487  
defun, 487  

pfBracketBar, 487  
called by npBracked, 410  
calls pfApplication, 487  
calls pfIdPos, 488  
calls tokPosn, 488  
defun, 487  

pfBreak, 488  
called by npBreak, 399  
calls pfTree, 488  
defun, 488  

pfBreak?, 488  
called by pf2Sex1, 531
INDEX

calls pfAbSynOp?, 488
defun, 488

pfBreakFrom, 488
called by pf2Sex1, 531
defun, 488

pfCharPosn, 465
called by ppos, 579
calls poCharPosn, 465
defun, 465

pfCheckArg, 471
called by pfCheckMacroOut, 470
calls pf0TupleParts, 471
calls pfCheckId, 471
calls pfListOf, 471
calls pfTuple?, 471
defun, 471

pfCheckId, 471
called by pfCheckArg, 471
called by pfCheckMacroOut, 470
calls npTrapForm, 471
calls pfId?, 471
defun, 471

pfCheckItOut, 469
called by npDef, 412
calls npTrapForm, 469
calls pf0TupleParts, 469
calls pfApplication?, 469
calls pfCollect1?, 469
calls pfCollectVariable1, 469
calls pfDefinition?, 469
calls pfFlattenApp, 469
calls pfId?, 469
calls pfListOf, 469
calls pfNothing, 469
calls pfTagged?, 469
calls pfTaggedExpr, 469
calls pfTaggedTag, 469
calls pfTaggedToTyped1, 469
calls pfTaggedToTyped, 469
calls pTransformArg, 469
calls pfTuple?, 469
calls pfTyped, 469
defun, 469

pfCheckMacroOut, 470
called by npMdef, 389
calls npTrapForm, 470
calls pfApplication?, 470
calls pfCheckArg, 470
calls pfCheckId, 470
calls pfFlattenApp, 470
calls pfId?, 470
defun, 470

pfCoerceto, 489
called by npCoerceTo, 444
calls pfTree, 489
defun, 489

pfCoerceto?, 489
called by pf2Sex1, 530
calls pfAbSynOp?, 489
defun, 489

pfCoercetoExpr, 489
called by pf2Sex1, 530
defun, 489

pfCoercetoType, 489
called by pf2Sex1, 530
defun, 489

pfCollect, 490
called by npExpress, 404
calls pfTree, 490
defun, 490

pfCollect1?, 472
called by pfCheckItOut, 469
called by pfFlattenApp, 472
called by pfTaggedToTyped1, 474
calls pfApplication?, 472
calls pfApplicationOp, 472
calls pfId?, 472
calls pfIdSymbol, 472
defun, 472

pfCollect2Sex, 544
called by pf2Sex1, 530
calls loopIters2Sex, 544
calls pf2Sex1, 544
calls pfCollectBody, 544
calls pfCollectIterators, 544
calls pfParts, 544
defun, 544

pfCollect?, 490
called by pf2Sex1, 530
called by pfCollectArgTran, 547
calls pfAbSynOp?, 490
defun, 490
defun, 493
pfExit?, 493
    called by pf2Sex1, 530
calls pfAbSynOp?, 493
defun, 493
pfExitCond, 493
    called by pf2Sex1, 530
defun, 493
pfExitExpr, 494
    called by pf2Sex1, 530
defun, 494
pfExport, 494
    called by npExport, 395
calls pfTree, 494
defun, 494
pfExpression, 494
    called by pfSymb, 481
calls ifcar, 494
calls pfLeaf, 494
defun, 494
pfFileName, 466
    called by ppos, 579
calls poFileName, 466
defun, 466
pfFirst, 494
    called by pfLam, 502
called by pfSourceStok, 473
defun, 494
pfFix, 495
    called by npFix, 390
calls pfApplication, 495
calls pfId, 495
defun, 495
pfFlattenApp, 471
    called by pfCheckItOut, 469
called by pfCheckMacroOut, 470
called by pfFlattenApp, 472
calls pfApplication?, 471
calls pfApplicationArg, 472
calls pfApplicationOp, 472
calls pfCollect1?, 472
calls pfFlattenApp, 472
defun, 471
pfForin, 496
    called by npForIn, 402
calls pfTree, 496
defun, 496
pfForin?, 496
    called by pf2Sex1, 530
calls pfAbSynOp?, 496
defun, 496
pfForinLhs, 497
    called by pf0ForinLhs, 497
defun, 497
pfForinWhole, 497
    called by pf2Sex1, 530
defun, 497
pfFree, 495
    called by npFree, 398
calls pfTree, 495
defun, 495
pfFree?, 495
    called by pf2Sex1, 531
calls pfAbSynOp?, 495
defun, 495
pfFreeItems, 496
    called by pf0FreeItems, 496
defun, 496
pfFromDom, 497
    called by npFromdom1, 428
called by npFromdom, 427
calls pfApplication?, 497
calls pfApplicationArg, 497
calls pfApplicationOp, 497
calls pfApplication, 497
calls pfFromdom, 497
defun, 497
pfFromdom, 498
    called by npFromDom, 497
calls pfTree, 498
defun, 498
pfFromdom?, 498
    called by pf2Sex1, 530
calls pfAbSynOp?, 498
defun, 498
pfFromdomDomain, 499
    called by pf2Sex1, 530
defun, 499
pfFromdomWhat, 498
    called by pf2Sex1, 530
defun, 498
pfGlobalLinePosn, 465
called by syIgnoredFromTo, 416
calls poGlobalLinePosn, 465
defun, 465
pfHide, 499
called by npAngleBared, 411
calls pfTree, 499
defun, 499
pfId, 476
called by pfFix, 495
called by pfSuch, 474
called by pfTaggedToTyped, 519
calls pfLeaf, 476
defun, 476
pfId?, 476
called by mac0MLambdaApply, 452
called by mac0SubstituteOuter, 459
called by macExpand, 451
called by macMacro, 458
called by pfCheckId, 471
called by pfCheckItOut, 469
called by pfCheckMacroOut, 470
called by pfCollect1?, 472
called by pfSexpr, strip, 480
called by pfTaggedToTyped, 519
calls pfAbSynOp?, 476
defun, 476
pfIdPos, 476
called by pfBraceBar, 487
called by pfBrace, 487
called by pfBracketBar, 488
called by pfBracket, 487
calls pfLeaf, 476
defun, 476
pfIdSymbol, 477
called by macId, 455
called by macLambdaParameterHandling, 460
called by macMacro, 458
called by macSubstituteId, 460
called by pf2Sex1, 530
called by pfCollect1?, 472
called by pfInfApplication, 501
called by pfSexpr, strip, 480
calls tokPart, 477
defun, 477
pff, 499
called by npElse, 421
called by pffThenOnly, 500
calls pfTree, 499
defun, 499
pff?, 499
called by pf2Sex1, 530
calls pfAbSynOp?, 499
defun, 499
pffCond, 500
called by pf2Sex1, 530
called by pfTweakIf, 520
defun, 500
pffElse, 500
called by pf2Sex1, 530
called by pfTweakIf, 520
defun, 500
pffThen, 500
called by pf2Sex1, 530
called by pfTweakIf, 520
defun, 500
pffThenOnly, 500
called by npElse, 421
calls pff, 500
calls pfNothing, 500
defun, 500
pffImmediate?
called by ppos, 579
pffImport, 501
called by npImport, 405
calls pfTree, 501
defun, 501
pffInfApplication, 501
called by npInterval, 424
called by npLeftAssoc, 431
called by npRightAssoc, 430
called by npSynthetic, 423
called by pfSuch, 474
called by pfTaggedToTyped, 520
calls pfAnd, 501
calls pfApplication, 501
calls pfIdSymbol, 501
calls pfListOf, 501
calls pfOr, 501
calls pfTuple, 501
defun, 501
pffinline, 502
<table>
<thead>
<tr>
<th>Function</th>
<th>Called by</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>pfIterate</td>
<td>pfTree, npIterate</td>
<td>502, 399</td>
</tr>
<tr>
<td>pfIterate?</td>
<td>pfTree, npIterate</td>
<td>501, 399</td>
</tr>
<tr>
<td>pfLam</td>
<td>pfLambda, npLam</td>
<td>502, 373</td>
</tr>
<tr>
<td>pfLambda</td>
<td>pfLambda, npLambda</td>
<td>503, 373</td>
</tr>
<tr>
<td>pfLambda2Sex</td>
<td>pfLambda, npLambda</td>
<td>503, 373</td>
</tr>
<tr>
<td>pfLambda?</td>
<td>pfLambda, npLambda, 546</td>
<td></td>
</tr>
<tr>
<td>pfLambdaBody</td>
<td>pfLambda, npLambda</td>
<td>503, 373</td>
</tr>
<tr>
<td>pfLambdaRets</td>
<td>pfLambda, npLambda</td>
<td>503, 373</td>
</tr>
<tr>
<td>pfLeaf</td>
<td>pfLeaf, npLeaf</td>
<td>477, 477</td>
</tr>
<tr>
<td>pfLeaf?</td>
<td>pfLeaf, npLeaf</td>
<td>477, 477</td>
</tr>
<tr>
<td>pfLeafPosition</td>
<td>pfLeaf, npLeaf</td>
<td>478, 478</td>
</tr>
</tbody>
</table>

---

**Index**

- called by npInline, 398
- calls pfTree, 502
defun, 502
- pfIterate, 501
called by npIterate, 399
calls pfTree, 501
defun, 501
- pfIterate?, 501
called by np2Sex1, 531
calls pfAbSynOp?, 501
defun, 501
- pfLam, 502
called by npLambda, 373
calls pfAbSynOp?, 502
calls pfFirst, 502
calls pfLambda, 502
calls pfNothing, 502
calls pfSecond, 502
defun, 502
- pfLambda, 503
called by pfLam, 502
called by pfPushBody, 479
calls pfTree, 503
defun, 503
- pfLambda2Sex, 548
called by pf2Sex1, 531
calls pfLambda, 502
defun, 503, 548
- pfLambda?, 503
called by mac0SubstituteOuter, 459
called by macExpand, 450
called by macLambdaParameterHandling, 460
called by pf2Sex1, 531
called by pfLambdaTrans, 546
calls pfAbSynOp?, 503
defun, 503
- pfLambdaArgs, 504
called by pf0LambdaArgs, 504
defun, 504
- pfLambdaBody, 503
called by pfLambdaTrans, 546
defun, 503
- pfLambdaRets, 503
called by pfLambdaTrans, 546
defun, 503
- pfLambdaTran, 546
called by pfDefinition2Sex, 545
called by pfLambda2Sex, 548
calls pf0LambdaArgs, 546
calls pf2Sex1, 546
calls pfCollectArgTrans, 546
calls pfLambda?, 546
calls pfLambdaBody, 546
calls pfLambdaRets, 546
calls pfNothing?, 546
calls pfTyped?, 546
calls pfTypedId, 546
calls pfTypedType, 546
calls systemError, 546
defun, 546
- pfLeaf, 477
called by macLambdaParameterHandling, 460
called by pfCopyWithPos, 466
called by pfDocument, 476
called by pfExpression, 494
called by pfIdPos, 476
called by pfId, 476
called by pfSymbol, 481
calls ifcar, 477
calls pfNoPosition, 477
calls tokConstruct, 477
defun, 477
- pfLeaf?, 477
called by mac0SubstituteOuter, 459
called by macLambdaParameterHandling, 459
called by pfCopyWithPos, 466
called by pfMapParts, 466
called by pfSexpr, strip, 480
called by pfSourcePosition, 468
called by pfSourceStok, 473
called by pfSymb, 481
calls pfAbSynOp, 477
defun, 477
- pfLeafPosition, 478
called by macLambdaParameterHandling, 460
called by pfSourcePosition, 468
calls tokPosn, 478
defun, 478
pfLeafToken, 478
called by pfLiteral2Sex, 534
calls tokPart, 478
defun, 478
pfLhsRule2Sex, 549
called by pfRule2Sex, 548
calls pf2Sex1, 549
uses $insideRule, 549
defun, 549
pfLinePosn, 465
called by ppEns, 579
calls poLinePosn, 465
defun, 465
pfListOf, 475
called by npAssignVariable, 442
called by npBpileDefinition, 412
called by npEnclosed, 436
called by npExpress, 404
called by npListing, 379
called by npParse, 365
called by npRecoverTrap, 415
called by npSigItemlist, 379
called by npTypeVariable, 380
called by npVariable, 438
called by pfCheckArg, 471
called by pfCheckItOut, 469
called by pfEnSequence, 493
called by pfInfApplication, 501
called by pfLoop, 505
called by pfLp, 506
called by pfSequenceToList, 469
called by pfTransformArg, 474
called by pfTupleListOf, 522
called by pfTweakIf, 520
called by pfUnSequence, 523
calls pfTree, 475
defun, 475
pfLiteral2Sex, 534
called by pf2Sex1, 530
calls float2Sex, 534
calls keyedSystemError, 534
calls pfLeafToken, 534
calls pfLiteralClass, 534
calls pfLiteralString, 534
calls pfSymbolSymbol, 534
uses $insideRule, 534
defun, 534
pfLiteral?, 478
called by pf2Sex1, 530
called by pfSexpr,strip, 480
calls pfAbSynOp, 478
defun, 478
pfLiteralClass, 478
called by pfLiteral2Sex, 534
calls pfAbSynOp, 478
defun, 478
pfLiteralString, 479
called by pfLiteral2Sex, 534
called by pfSexpr,strip, 480
calls tokPart, 479
defun, 479
pfLocal, 504
called by npLocal, 398
calls pfTree, 504
defun, 504
pfLocal?, 504
called by pf2Sex1, 531
calls pfAbSynOp?, 504
defun, 504
pfLocalItems, 505
called by pf0LocalItems, 505
defun, 505
pfLoop, 505
called by npLoop1, 505
called by pfLp, 506
calls pfTree, 505
defun, 505
pfLoop1, 505
called by npLoop, 399
calls pfDo, 505
calls pfListOf, 505
calls pfLoop, 505
defun, 505
pfLoop?, 506
called by pf2Sex1, 530
calls pfAbSynOp?, 506
defun, 506
pfLoopIterators, 506
defun, 506
pfLp, 506
called by npLoop, 399
calls pfDo, 506
calls pfListOf, 506
calls pfLoop, 506
defun, 506
pfMacro, 507
calledby macMacro, 458
calledby npMdef, 389
calls pfTree, 507
defun, 507
pfMacro?, 507
calledby macExpand, 451
calledby pf2Sex1, 531
calls pfAbSynOp?, 507
defun, 507
pfMacroLhs, 507
calledby macMacro, 457
defun, 507
pfMacroRhs, 507
calledby macMacro, 458
defun, 507
pfMapParts, 466
calledby macApplication, 451
calledby macExpand, 451
calledby macLambda,mac, 457
calledby macWhere,mac, 456
calls pfAbSynOp, 467
calls pfLeaf?, 466
calls pfParts, 466
calls pfTree, 467
defun, 466
pfMLambda, 508
calledby pfPushMacroBody, 473
calls pfTree, 508
defun, 508
pfMLambda?, 508
calledby macApplication, 451
calledby macLambdaParameterHandling, 460
calledby macMacro, 458
calledby pf2Sex1, 531
calls pfAbSynOp?, 508
defun, 508
pfMLambdaArgs, 508
defun, 508
pfMLambdaBody, 509
calledby mac0GetName, 454
calledby mac0MLambdaApply, 452
defun, 509
pfname, 316
calledby porigin, 313
calledby thefname, 316
calls PathnameString, 316
defun, 316
pfNoPosition, 636
calledby pfLeaf, 477
calledby tokPosn, 635
calls poNoPosition, 636
defun, 636
pfNoPosition?, 634
calledby ppos, 579
calledby tokConstruct, 633
calls poNoPosition?, 634
defun, 634
pfNot?, 509
calledby pf2Sex1, 531
calls pfAbSynOp?, 509
defun, 509
pfNotArg, 509
calledby pf2Sex1, 531
defun, 509
pfNothing, 475
calledby macMacro, 458
calledby npAdd, 383
calledby npBreak, 399
calledby npDefaultValue, 419
calledby npIterate, 399
calledby npLocalDecl, 397
calledby npPileBracketed, 413
calledby npPrimary2, 382
calledby npQualType, 406
calledby npReturn, 403
calledby npSignature, 378
calledby npVariableName, 438
calledby npWith, 375
calledby pfAdd, 482
calledby pfCheckItOut, 469
calledby pfThenOnly, 500
calledby pfLam, 502
calledby pfPushBody, 479
calledby pfReturnNoName, 515
calledby pfTaggedToTyped1, 474
calledby pfTaggedToTyped, 519
calls pfTree, 475
defun, 475
pfNothing?, 475
called by macMacro, 458
called by pf2Sex1, 530
called by pfLambdaTran, 546
called by pfTweakIf, 520
calls pfAbSynOp?, 475
defun, 475
pfNothing?, 475
called by macro, 458
called by pf2Sex1, 530
called by pfLambdaTran, 546
called by pfTweakIf, 520
calls pfAbSynOp?, 475
defun, 475
pfNothing?, 475
called by macro, 458
called by pf2Sex1, 530
called by pfLambdaTran, 546
called by pfTweakIf, 520
calls pfAbSynOp?, 475
defun, 475
pfNothing?, 475
called by macro, 458
called by pf2Sex1, 530
called by pfLambdaTran, 546
called by pfTweakIf, 520
calls pfAbSynOp?, 475
defun, 475
pfParen, 511
called by npParened, 410
defun, 511
pfParts, 479
called by macro SubstituteOuter, 459
called by macro LambdaParameterHandling, 460
called by npDefaultDecl, 394
called by npLocalDecl, 397
called by npDefaultItem, 393
called by npLocalItem, 396
called by npQualTypelist, 405
called by npSigDecl, 381
called by npSigItemlist, 379
called by p0AssignLhsItems, 486
called by p0DefinitionLhsItems, 492
called by p0ForInLhs, 497
called by p0FreeItems, 496
called by p0LambdaArgs, 504
called by p0LocalItems, 505
called by p0LoopIterators, 506
called by p0MLambdaArgs, 508
called by p0SequenceArgs, 517
called by p0TupleParts, 523
called by p0WhereContext, 525
called by p0Collect2Sex, 544
called by p0CollectArgTran, 547
called by p0CopyWithPos, 466
called by p0MapParts, 466
called by p0Sexpr, 480
called by p0SourcePosition, 486
called by p0SourceStok, 473
called by p0WDec, 524
defun, 479
pfPile, 479
called by npPileBracketed, 413
defun, 479
pfPretend, 511
called by npPretend, 414
called by p0Tree, 510
defun, 511
pfPretend?, 512
called by p0Tree, 510
defun, 511
pfPretend?, 512
called by p0Tree, 510
defun, 511
pfPretend?, 512
called by p0Tree, 510
defun, 511
pfPretend?, 512
called by p0Tree, 510
defun, 511
pfPretend?, 512
called by p0Tree, 510
defun, 511
pfPretendExpr, 512
called by pf2Sex1, 530
defun, 512
pfPretendType, 512
called by pf2Sex1, 530
defun, 512
pfPrintSrcLines
called by displayParserMacro, 736
pfPushBody, 479
called by npDef, 412
called by pfPushBody, 479
calls pMLambda, 479
calls pNothing, 479
calls pfPushBody, 479
defun, 479
pfPushMacroBody, 473
called by npMidef, 389
called by pfPushMacroBody, 473
calls pMLambda, 473
calls pPushMacroBody, 473
defun, 473
pfQualType, 512
called by npQualType, 406
calls pfTree, 512

defun, 512
pfRestrict, 513
called by npRestrict, 445
calls pfTree, 513
defun, 513
pfRestrict?, 513
called by pf2Sex1, 531
calls pfAbSynOp?, 513
defun, 513
pfRestrictExpr, 513
called by pf2Sex1, 531
defun, 513
pfRestrictType, 513
called by pf2Sex1, 531
defun, 513
pfRetractTo, 514
called by npColonQuery, 444
calls pfTree, 514
defun, 514
pfReturn, 514
called by npReturn, 403
called by pfReturnNoName, 515
calls pfTree, 514
defun, 514
pfReturn?, 514
called by pf2Sex1, 531
calls pfAbSynOp?, 514
defun, 514
pfReturnExpr, 514
called by pf2Sex1, 531
defun, 514
pfReturnNoName, 515
called by npReturn, 403
calls pfNothing, 515
calls pfReturn, 515
defun, 515
pfReturnTyped, 515
called by npLambda, 373
calls pfTree, 515
defun, 515
pfRhsRule2Sex, 549
called by pfRule2Sex, 548
calls pf2Sex1, 549
uses $insideRule, 549
defun, 549
pfRule, 515
called by npSingleRule, 418
calls pfTree, 515
defun, 515
pfRule2Sex, 548
called by pf2Sex1, 531
calls pfLhsRule2Sex, 548
calls pfRhsRule2Sex, 548
calls pfRuleLhsItems, 548
calls pfRuleRhs, 548
calls ruleLhsTran, 548
calls rulePredicateTran, 548
uses $multiVarPredicateList, 548
uses $predicateList, 548
uses $quotedOpList, 548
defun, 548
pfRule?, 516
called by pf2Sex1, 531
calls pfAbSynOp?, 516
defun, 516
pfRuleLhsItems, 516
called by pfRule2Sex, 548
defun, 516
pfRuleRhs, 516
    called by pfRule2Sex, 548
defun, 516
pfSecond, 516
    called by pfLam, 502
defun, 516
pfSequence, 517
    called by npBPileDefinition, 412
called by pfEnSequence, 493
calls pTree, 517
defun, 517
pfSequence2Sex, 540
    called by pf2Sex1, 530
calls pf0SequenceArgs, 540
calls pf2Sex1, 540
uses $insideSEQ, 540
defun, 540
pfSequence2Sex0, 540
    called by pfSequence2Sex0, 541
calls pfSequence2Sex0, 541
defun, 540
pfSequence?, 517
    called by pf2Sex1, 530
called by pfSequenceToList, 468
called by pfUnSequence, 523
calls pfAbSynOp?, 517
defun, 517
pfSequenceArgs, 517
    called by pf0SequenceArgs, 517
called by pfSequenceToList, 468
defun, 517
pfSequenceToList, 468
    called by npDefinition, 391
calls pfListOf, 469
calls pfSequence?, 468
calls pfSequenceArgs, 468
defun, 468
pfSexpr, 480
    called by pfSymb, 481
calls pfSexpr,strip, 480
defun, 480
pfSexpr,strip, 480
    called by pfSexpr,strip, 480
called by pfSexpr, 480
calls pf0TupleParts, 480
calls pfAbSynOp, 480
calls pfApplication?, 480
calls pfApplicationArg, 480
calls pfApplicationOp, 480
calls pfId?, 480
calls pfIdSymbol, 480
calls pfLeaf?, 480
calls pfLiteral?, 480
calls pfLiteralString, 480
calls pfParts, 480
calls pfSexpr,strip, 480
calls pfTuple?, 480
calls tokPart, 480
defun, 480
pfSourcePosition, 468
    called by mac0ExpandBody, 453
called by mac0MLambdaApply, 452
called by macId, 455
called by macMacro, 458
called by pfSourcePosition, 468
calls pfLeaf?, 468
calls pfLeafPosition, 468
calls pfParts, 468
calls pfSourcePosition, 468
calls poNoPosition?, 468
uses $nopos, 468
defun, 468
pfSourceStok, 473
    called by npTrapForm, 437
called by pfSourceStok, 473
calls pfFirst, 473
calls pfLeaf?, 473
calls pfParts, 473
calls pfSourceStok, 473
defun, 473
pfSpread, 469
    called by npDefaultDecl, 394
called by npLocalDecl, 397
called by npSigDecl, 381
calls pfTyped, 469
defun, 469
pfSuch, 474
    called by pfCollectVariable1, 472
called by pfTaggedToTyped, 519
calls pfId, 474
calls pfInfApplication, 474
defun, 474
INDEX

pfSuchthat, 518
  called by npSuchThat, 401
calls pfTree, 518
defun, 518

pfSuchthat2Sex, 537
called by pfApplication2Sex, 536
calls p0TupleParts, 537
calls pf2Sex1, 537
calls pf2Sex, 537
uses $predicateList, 537
defun, 537

pfSuchthat?, 518
called by pf2Sex1, 531
calls pfAbSynOp?, 518
defun, 518

pfSuchthatCond, 518
called by pf2Sex1, 531
defun, 518

pfSymb, 481
  called by npConstTok, 409
called by npDDInfKey, 432
called by npInfApplication, 384
calls ifcar, 481
calls pfExpression, 481
calls pfLeaf?, 481
calls pfSexpr, 481
calls pfSymbol, 481
calls tokPart, 481
defun, 481

pfSymbol, 481
called by pfSymb, 481
calls ifcar, 481
calls pfLeaf, 481
defun, 481

pfSymbol?, 482
called by pf2Sex1, 530
called by pfOp2Sex, 538
calls pfAbSynOp?, 482
defun, 482

pfSymbolSymbol, 482
called by pf2Sex1, 530
called by pfLiteral2Sex, 534
calls tokPart, 482
defun, 482

pfTagged, 518
called by npTagged, 442
calls pfTree, 518
defun, 518

pfTagged?, 519
called by pf2Sex1, 530
called by pfCheckItOut, 469
called by pfTaggedToTyped, 519
calls pfAbSynOp?, 519
defun, 519

pfTaggedExpr, 519
called by pf2Sex1, 530
called by pfCheckItOut, 469
called by pfTaggedToTyped, 519
defun, 519

pfTaggedTag, 519
called by pf2Sex1, 530
called by pfCheckItOut, 469
called by pfTaggedToTyped, 519
defun, 519

pfTaggedToTyped, 519
called by pfCheckItOut, 469
called by pfCollectVariable1, 472
called by pfTaggedToTyped1, 474
calls pfId?, 519
calls pfId, 519
calls pfInfApplication, 520
calls pfNothing, 519
calls pfSuch, 519
calls pfTagged?, 519
calls pfTaggedExpr, 519
calls pfTaggedTag, 519
calls pfTyped, 519
defun, 519

pfTaggedToTyped1, 474
called by pfCheckItOut, 469
called by pfTransformArg, 474
calls pfCollect1?, 474
calls pfCollectVariable1, 474
calls pfDefinition?, 474
calls pfNothing, 474
calls pfTaggedToTyped, 474
calls pfTyped, 474
defun, 474

pfTransformArg, 474
called by pfCheckItOut, 469
calls p0TupleParts, 474
calls pfListOf, 474
calls pfTaggedToTyped1, 474
calls pfTuple?, 474
defun, 474
pfTree, 482
called by pfAdd, 482
called by pfAnd, 483
called by pfApplication, 483
called by pfAssign, 485
called by pfAttribute, 483
called by pfBreak, 488
called by pfCoerceTo, 489
called by pfCollect, 490
called by pfCopyWithPos, 466
called by pfDefinition, 491
called by pfDo, 492
called by pfExit, 493
called by pfExport, 494
called by pfForin, 496
called by pfFree, 495
called by pfFromdom, 498
called by pfHide, 499
called by pfIf, 499
called by pfImport, 501
called by pfInline, 502
called by pfIterate, 501
called by pfLambda, 503
called by pfListOf, 475
called by pfLocal, 504
called by pfLoop, 505
called by pfMLambda, 508
called by pfMacro, 507
called by pfMapParts, 467
called by pfNothing, 475
called by pfNoValue, 509
called by pfOr, 510
called by pfPretend, 511
called by pfQualType, 512
called by pfRestrict, 513
called by pfRestractTo, 514
called by pfReturnTyped, 515
called by pfReturn, 514
called by pfRule, 515
called by pfSequence, 517
called by pfSuchthat, 518
called by pfTagged, 518
called by pfTuple, 522
called by pfTweakIf, 520
called by pfTyped, 520
called by pfTyping, 521
called by pfWDeclare, 524
called by pfWhere, 524
called by pfWhile, 525
called by pfWith, 526
called by pfWrong, 526
defun, 482
pfTuple, 522
called by npEnclosed, 436
called by npEnSequence, 493
called by npInfApplication, 501
called by npTupleListOf, 522
calls pfTree, 522
defun, 522
pfTuple?, 522
called by np0FlattenSyntacticTuple, 467
called by np2Sex1, 530
called by npApplication2Sex, 536
called by npCheckArg, 471
called by npCheckItOut, 469
called by npSexpr,strip, 480
called by npTransformArg, 474
calls pfAbSynOp?, 522
defun, 522
pfTupleListOf, 522
called by npTuple, 370
calls pfListOf, 522
calls pfTuple, 522
defun, 522
pfTupleParts, 523
called by np0TupleParts, 523
defun, 523
pfTweakIf, 520
called by npWConditional, 420
calls pfIfCond, 520
calls pfIfElse, 520
calls pfIfThen, 520
calls pfListOf, 520
calls pfNothing?, 520
calls pfTree, 520
defun, 520
pfTyped, 520
called by npDecl, 439
called by npVariableName, 438
INDEX

1799
called by pfCheckItOut, 469
called by pfCollectVariable1, 472
called by pfSpread, 469
called by pfTaggedToTyped1, 474
called by pfTaggedToTyped, 519
calls pfTree, 520
defun, 520
pfTyped?, 521
called by pf2Sex1, 531
called by pfLambdaTran, 546
calls pfAbSynOp?, 521
defun, 521
pfTypedId, 521
called by macLambdaParameterHandling, 460
called by pf2Sex1, 531
called by pfCollectVariable1, 472
called by pfLambdaTran, 546
defun, 521
pfTyping, 521
called by npTyping, 393
calls pfTree, 521
defun, 521
pfUnSequence, 523
called by npCategoryL, 377
called by npDefaultItemlist, 393
called by npLocalItemlist, 396
called by npQualTypelist, 405
calls pf0SequenceArgs, 523
calls pfAppend, 523
calls pfListOf, 523
calls pfSequence?, 523
defun, 523
pfWDec, 523
called by npSignature, 378
calls pfParts, 524
calls pfWDeclare, 523
defun, 523
pfWDeclare, 524
called by pfWDec, 523
calls pfTree, 524
defun, 524
pfWhere, 524
called by npLetQualified, 391
called by npQualified, 371
calls pfTree, 524
defun, 524
pfWhere?, 524
called by macExpand, 450
called by pf2Sex1, 531
calls pfAbSynOp?, 524
defun, 524
pfWhereContext, 525
called by pf0WhereContext, 525
defun, 525
pfWhereExpr, 525
called by pf2Sex1, 531
defun, 525
pfWhile, 525
called by npWhile, 402
calls pfTree, 525
defun, 525
pfWhile?, 526
called by pf2Sex1, 530
calls pfAbSynOp?, 526
defun, 526
pfWhileCond, 526
called by pf2Sex1, 530
defun, 526
pfWith, 526
called by npWith, 375
calls pfTree, 526
defun, 526
pfWrong, 526
called by npParse, 365
called by npRecoverTrap, 415
calls pfTree, 526
defun, 526
pfWrong?, 527
called by pf2Sex1, 531
calls pfAbSynOp?, 527
defun, 527
phInterpret, 292
calls intInterpretPform, 292
calls ncEltQ, 292
calls ncPutQ, 292
defun, 292
phIntReportMsgs, 291
calls ncElitQ, 291
calls ncPutQ, 291
calls processMsgList, 291
uses $erMsgToss, 291
defun, 291
phMacro, 449
calls macroExpanded, 450
calls ncElitQ, 450
calls ncPutQ, 450
calls pforn, 450
defun, 449
phParse, 291
calls ncPutQ, 291
defun, 291
pileCforest, 562
calledby insertpile, 557
calledby pileCtree, 562
calls enPile, 562
calls separatePiles, 562
calls tokPart, 562
defun, 562
pileColumn, 559
calledby eqpileTree, 561
calledby pileTree, 559
calls tokPosn, 559
defun, 559
pileCtree, 562
calledby pileForests, 560
calls dqAppend, 562
calls pileCforest, 562
defun, 562
pileForest, 560
calledby pileForests, 559
calls pileForest1, 560
calls pileTree, 560
defun, 560
pileForest1, 560
calledby pileForest1, 561
calledby pileForest, 560
calls eqpileTree, 560
calls pileForest1, 561
defun, 560
pileForests, 559
calledby eqpileTree, 561
calledby pileForests, 560
calledby pileTree, 559
calls npNull, 560
calls pileCtree, 560
calls pileForest, 560
calls pileForest, 559
defun, 559
pilePlusComment, 558
calledby insertpile, 557
calledby pilePlusComments, 558
calls tokType, 558
defun, 558
pilePlusComments, 558
calledby insertpile, 557
calledby pilePlusComments, 558
calls npNull, 558
calls pilePlusComments, 558
calls pilePlusComment, 558
defun, 558
pileTree, 559
calledby insertpile, 557
calledby pileForest, 560
calls npNull, 559
calls pileColumn, 559
calls pileForests, 559
defun, 559
placep, 1227
calledby unwritable?, 848
calledby writify,writifyInner, 849
uses *read-place-holder*, 1227
defun, 1227
pluralize
calledby dbConsHeading, 1553
calledby dbShowConditions, 1552
calledby kePageDisplay, 1512
pluralSay

calledby kePage, 1510
pmDontQuote?, 539
calledby pfOp2Sex, 538
defun, 539
pmTransFilter
calledby dbShowCons, 1545
calledby koaPageFilterByName, 1538
pname, 1195
calledby clearCmdParts, 773
calledby coerceTraceArg2E, 62
calledby coerceTraceFunValue2E, 64
INDEX

calledby displayType, 732
calledby displayValue, 731
calledby fixObjectForPrinting, 728
calledby gensymInt, 859
calledby getAliasIfTracedMapParameter, 75
calledby getStFromMsg, 577
calledby hasOption, 723
calledby isSharpVarWithNum, 1063
calledby kcnPage, 1526
calledby letPrint2, 1065
calledby letPrint3, 1066
calledby letPrint, 1062
calledby mac0InfiniteExpansion, name, 454
calledby newHelpSpad2Cmd, 813
calledby npMissing, 376
calledby reportUndo, 44
calledby rwrite, 846, 848
calledby selectOption, 751
calledby setFortDir, 930
calledby setFortTmpDir, 928
calledby setOutputCharacters, 971
calledby stupidIsSpadFunction, 1070
calledby undo, 36
defun, 1195
poCharPosn, 599
calledby compareposns, 592
calledby pfCharPosn, 465
calledby posPointers, 600
calledby processChPosesForOneLine, 598
calledby putFTText, 601
defun, 599
poFileName, 582
calledby decideHowMuch, 581
calledby pfFileName, 466
calls lnFileName, 582
calls poGetLineObject, 583
defun, 582
poGetLineObject, 583
calledby poFileName, 583
calledby poGlobalLinePosn, 296
calledby poLinePosn, 583
calledby poPosImmediate?, 582
defun, 583
poGlobalLinePosn, 296
calledby compareposns, 592
calledby listDecideHowMuch, 583
calledby makeMsgFromLine, 593
calledby nclloopDQlines, 296
calledby pfGlobalLinePosn, 465
calledby processMsgList, 591
calledby thisPosIsEqual, 596
calledby thisPosIsLess, 596
calls lnGlobalNum, 296
calls ncBug, 296
calls poGetLineObject, 296
defun, 296
poLinePosn, 583
calledby decideHowMuch, 582
calledby makeMsgFromLine, 593
calledby pfLinePosn, 465
calls lnLocalNum, 583
calls poGetLineObject, 583
defun, 583
poNopos?, 582
calledby decideHowMuch, 581
calledby erMsgSep, 592
calledby listDecideHowMuch, 583
calledby poPosImmediate?, 582
calledby thisPosIsEqual, 596
calledby thisPosIsLess, 596
defun, 582
poNoPosition, 636
calledby pfNoPosition, 636
geno $nopos, 636
defun, 636
poNoPosition?, 635
calledby pfNoPosition?, 634
calledby pfSourcePosition, 468
calls eqcar, 635
defun, 635
poPosImmediate?, 582
calledby decideHowMuch, 581
calledby listDecideHowMuch, 583
calls lnImmediate?, 582
calls poGetLineObject, 582
calls poNopos?, 582
defun, 582
porigin, 313
calledby inclmsgFileCycle, 317
calledby ppos, 579
calls pfname, 313


\begin{verbatim}
defun, 313
posend, 352
called by scanW, 351
defun, 352
position
called by coerceBranch2Union, 698
called by dbSubConform, 1544
called by kArgPage, 1506
posPointers, 600
called by processChPosesForOneLine, 598
calls getMsgPos2, 600
calls getMsgPos, 600
calls ifcar, 600
calls insertPos, 600
calls poCharPosn, 600
uses getMsgFTTag, 600
defun, 600
poundsign
called by dewritify, dewritifyInner, 854
called by displaySetVariableSettings, 897
called by getTraceOptions, 52
called by isDomainOrPackage, 1061
called by newHelpSpad2Cmd, 813
called by set1, 1014
called by setOutputLibrary, 904
called by trace1, 48
called by traceReply, 81
poundsign
called by mkConform, 1532
called by reportUndo, 44
pp2Cols
called by filterAndFormatConstructors, 1096
ppos, 579
called by getPosStL, 578
calls pfCharPosn, 579
calls pfFileName, 579
calls pfImmediate?, 579
calls pfLinePosn, 579
calls pfNoPosition?, 579
calls porigin, 579
defun, 579
pquit, 869, 870
calls pquitSpad2Cmd, 870
defun, 870
manpage, 869
pquitSpad2Cmd, 870
called by pquit, 870
calls quitSpad2Cmd, 870
uses $quitCommandType, 870
defun, 870

defun, 313
defun, 352
called by scanW, 351
defun, 352
position
called by coerceBranch2Union, 698
called by dbSubConform, 1544
called by kArgPage, 1506
posPointers, 600
called by processChPosesForOneLine, 598
calls getMsgPos2, 600
calls getMsgPos, 600
calls ifcar, 600
calls insertPos, 600
calls poCharPosn, 600
uses getMsgFTTag, 600
defun, 600
poundsign
called by dewritify, dewritifyInner, 854
called by displaySetVariableSettings, 897
called by getTraceOptions, 52
called by isDomainOrPackage, 1061
called by newHelpSpad2Cmd, 813
called by set1, 1014
called by setOutputLibrary, 904
called by trace1, 48
called by traceReply, 81
poundsign
called by mkConform, 1532
called by reportUndo, 44
pp2Cols
called by filterAndFormatConstructors, 1096
ppos, 579
called by getPosStL, 578
calls pfCharPosn, 579
calls pfFileName, 579
calls pfImmediate?, 579
calls pfLinePosn, 579
calls pfNoPosition?, 579
calls porigin, 579
defun, 579
pquit, 869, 870
calls pquitSpad2Cmd, 870
defun, 870
manpage, 869
pquitSpad2Cmd, 870
called by pquit, 870
calls quitSpad2Cmd, 870
uses $quitCommandType, 870
defun, 870

\end{verbatim}
called by recordAndPrint, 282
calls makeLongSpaceString, 283
uses $collectOutput, 283
uses $interpreterTimedClasses, 283
uses $interpreterTimedNames, 284
defun, 283

printSynonyms, 745
called by npProcessSynonym, 745
called by synonymSpad2Cmd, 1037
called by whatSpad2Cmd, 1091
calls filterListOfStringsWithFn, 745
calls printLabeledList, 745
calls specialChar, 745
calls synonymsForUserLevel, 745
uses $CommandSynonymAlist, 745
uses $linelength, 746
defun, 745

printTypeAndTime, 284
called by recordAndPrint, 282
calls justifyMyType, 284
calls makeLongTimeString, 284
calls mkObjWrap, 284
calls msgText, 284
calls objMode, 284
calls qcar, 284
calls retract, 284
calls sameUnionBranch, 284
calls sayKeyedMsg, 284
uses $collectOutput, 284
uses $interpreterTimedClasses, 284
uses $interpreterTimedNames, 284
uses $outputLines, 284
uses $printTimeIfTrue, 284
defun, 284

processInteractive, 274
called by intInterpretPform, 292
called by kisValidType, 1530
called by topLevelInterpEval, 1530
calls clrhash, 274
calls initializeTimedNames, 274
calls npProcessSynonym, 745
calls processInteractive1, 274
calls qcar, 274
calls reportInstantiations, 274
calls updateHist, 274
calls writeHistModesAndValues, 274
uses $Coerce, 275
uses $ProcessInteractiveValue, 275
uses $StreamFrame, 275
defun, 275

prior-token
used by token-stack-show, 1117

probeName, 1130
defun, 1130

processChPosesForOneLine, 598
called by queueUpErrors, 594
calls getMsgFTTag?, 598
calls getMsgPos, 598
calls getMsgPrefix, 598
calls makeLeaderMsg, 598
calls poCharPosn, 598
calls posPointers, 598
calls putFTText, 598
calls setMsgPrefix, 598
calls size, 598
calls strconc, 598
uses $collectOutput, 598
uses $interpreterTimedClasses, 598
uses $interpreterTimedNames, 598
defun, 598

processInteractive, 274
called by intInterpretPform, 292
called by kisValidType, 1530
called by topLevelInterpEval, 1530
calls clrhash, 274
calls initializeTimedNames, 274
calls npProcessSynonym, 745
calls processInteractive1, 274
calls qcar, 274
calls reportInstantiations, 274
calls updateHist, 274
calls writeHistModesAndValues, 274
uses $Coerce, 275
uses $ProcessInteractiveValue, 275
uses $StreamFrame, 275
defun, 275

prior-token
used by token-stack-show, 1117

probeName, 1130
defun, 1130

processChPosesForOneLine, 598
called by queueUpErrors, 594
calls getMsgFTTag?, 598
calls getMsgPos, 598
calls getMsgPrefix, 598
calls makeLeaderMsg, 598
calls poCharPosn, 598
calls posPointers, 598
calls putFTText, 598
calls setMsgPrefix, 598
calls size, 598
calls strconc, 598
uses $collectOutput, 598
uses $interpreterTimedClasses, 598
uses $interpreterTimedNames, 598
defun, 598

processInteractive, 274
called by intInterpretPform, 292
called by kisValidType, 1530
called by topLevelInterpEval, 1530
calls clrhash, 274
calls initializeTimedNames, 274
calls processInteractive1, 274
calls processInteractive, 274
calls qcar, 274
calls reportInstantiations, 274
calls updateHist, 274
calls writeHistModesAndValues, 274
uses $Coerce, 275
uses $ProcessInteractiveValue, 275
uses $StreamFrame, 275
defun, 275

prior-token
used by token-stack-show, 1117

probeName, 1130
defun, 1130

processChPosesForOneLine, 598
called by queueUpErrors, 594
calls getMsgFTTag?, 598
calls getMsgPos, 598
calls getMsgPrefix, 598
calls makeLeaderMsg, 598
calls poCharPosn, 598
calls posPointers, 598
calls putFTText, 598
calls setMsgPrefix, 598
calls size, 598
calls strconc, 598
uses $collectOutput, 598
uses $interpreterTimedClasses, 598
uses $interpreterTimedNames, 598
defun, 598

processInteractive, 274
called by intInterpretPform, 292
called by kisValidType, 1530
called by topLevelInterpEval, 1530
calls clrhash, 274
calls initializeTimedNames, 274
calls npProcessSynonym, 745
calls processInteractive1, 274
calls qcar, 274
calls reportInstantiations, 274
calls updateHist, 274
calls writeHistModesAndValues, 274
uses $Coerce, 275
uses $ProcessInteractiveValue, 275
uses $StreamFrame, 275
defun, 275

prior-token
used by token-stack-show, 1117

probeName, 1130
defun, 1130

processChPosesForOneLine, 598
called by queueUpErrors, 594
calls getMsgFTTag?, 598
calls getMsgPos, 598
calls getMsgPrefix, 598
calls makeLeaderMsg, 598
calls poCharPosn, 598
calls posPointers, 598
calls putFTText, 598
calls setMsgPrefix, 598
calls size, 598
calls strconc, 598
uses $collectOutput, 598
uses $interpreterTimedClasses, 598
uses $interpreterTimedNames, 598
defun, 598

processInteractive, 274
called by intInterpretPform, 292
called by kisValidType, 1530
called by topLevelInterpEval, 1530
calls clrhash, 274
calls initializeTimedNames, 274
calls processInteractive1, 274
calls processInteractive, 274
calls qcar, 274
calls reportInstantiations, 274
calls updateHist, 274
calls writeHistModesAndValues, 274
uses $Coerce, 275
uses $ProcessInteractiveValue, 275
uses $StreamFrame, 275
defun, 275

prior-token
used by token-stack-show, 1117

probeName, 1130
defun, 1130
processInteractive, 277
calledby processInteractive, 274
calls interpretTopLevel, 277
calls objMode, 277
calls objValUnwrap, 277
calls recordAndPrint, 277
calls recordFrame, 277
calls startTimingProcess, 277
calls stopTimingProcess, 277
uses $InteractiveFrame, 277
uses $ProcessInteractiveValue, 277
uses $e, 277
defun, 277
processKeyedError, 575
calledby ncBug, 590
calledby ncHardError, 574
calledby ncSoftError, 574
calls CallerName, 575
calls getMsgKey, 575
calls getMsgPrefix?, 575
calls getMsgTag?, 575
calls msgInPr?, 575
calls msgOutputter, 575
calls sayBrightly, 575
uses $ncMsgList, 575
defun, 575
processMsgList, 591
calledby phIntReportMsgs, 291
calls erMsgSort, 591
calls getMsgPos, 591
calls listOutputter, 591
calls makeMsgFromLine, 591
calls poGlobalLinePosn, 591
calls queueUpErrors, 591
uses $noRepList, 591
uses $outputList, 591
defun, 591
processSynonymLine, 1039
calledby npProcessSynonym, 745
calledby synonymSpad2Cmd, 1037
calls processSynonymLine,removeKeyFromLine, 1039
defun, 1039
processSynonyms, 257
calledby doSystemCommand, 718
calledby intProcessSynonyms, 257
calledby processSynonyms, 257
calls concat, 257
calls lassoc, 257
calls processSynonyms, 257
calls rplacstr, 257
calls size, 257
calls strconc, 257
calls string2id-n, 257
calls strpos, 257
calls substring, 257
uses $CommandSynonymAlist, 257
uses line, 257
defun, 257
protectedEVAL, 271
calledby serverReadLine, 269
calls resetStackLimits, 271
calls sendHTErrorLimits, 271
defun, 271
prTraceNames, 80
calls exit, 80
calls prTraceNames,fn, 80
calls seq, 80
uses /tracenames, 80
defun, 80
prTraceNames,fn, 79
calledby prTraceNames, 80
calls devaluate, 80
calls exit, 80
calls isDomainOrPackage, 80
calls qcar, 80
calls qcdr, 80
calls seq, 79
defun, 79
<table>
<thead>
<tr>
<th>calls concat, 1052</th>
<th></th>
<th>calls putIntSymTab, 832</th>
</tr>
</thead>
<tbody>
<tr>
<td>calls sayBrightly, 1052</td>
<td></td>
<td>calls recordNewValue, 832</td>
</tr>
<tr>
<td>uses /spaceslist, 1052</td>
<td></td>
<td>calls recordOldValue, 832</td>
</tr>
<tr>
<td>defun, 1052</td>
<td></td>
<td>uses $HiFiAccess, 832</td>
</tr>
<tr>
<td>ptimers, 1052</td>
<td></td>
<td>defun, 832</td>
</tr>
<tr>
<td>calledby trace1, 48</td>
<td></td>
<td>putIntSymTab</td>
</tr>
<tr>
<td>calls bright, 1052</td>
<td></td>
<td>calledby putHist, 832</td>
</tr>
<tr>
<td>pcall concat, 1052</td>
<td></td>
<td>putTarget</td>
</tr>
<tr>
<td>calls float, 1052</td>
<td></td>
<td>calledby evaluateType1, 1080</td>
</tr>
<tr>
<td>calls quotient, 1052</td>
<td></td>
<td>calledby interpret1, 279</td>
</tr>
<tr>
<td>calls sayBrightly, 1052</td>
<td></td>
<td>pvarCondList, 1434</td>
</tr>
<tr>
<td>uses /timerlist, 1052</td>
<td></td>
<td>defun, 1434</td>
</tr>
<tr>
<td>defun, 1052</td>
<td></td>
<td>pvarCondList1, 1434</td>
</tr>
<tr>
<td></td>
<td></td>
<td>defun, 1434</td>
</tr>
<tr>
<td>punctuation?, 341</td>
<td></td>
<td>pvarPredTran, 552</td>
</tr>
<tr>
<td>calledby scanToken, 337</td>
<td></td>
<td>calledby rulePredicateTran, 550</td>
</tr>
<tr>
<td>defun, 341</td>
<td></td>
<td>defun, 552</td>
</tr>
<tr>
<td>put, 1189</td>
<td></td>
<td>pvarsOfPattern, 1435</td>
</tr>
<tr>
<td>defun, 1189</td>
<td></td>
<td>defun, 1435</td>
</tr>
<tr>
<td>putalist</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calledby interpFunctionDepAlists, 737</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calledby npProcessSynonym, 745</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calledby synonymSpad2Cmd, 1037</td>
<td></td>
<td></td>
</tr>
<tr>
<td>putDatabaseStuff, 587</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calledby msgCreate, 570</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calls getMsgInfoFromKey, 587</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calls setMsgText, 587</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calls setMsgUnforcedAttrList, 587</td>
<td></td>
<td></td>
</tr>
<tr>
<td>defun, 587</td>
<td></td>
<td></td>
</tr>
<tr>
<td>putFTText, 601</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calledby processChPosesForOneLine, 598</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calls getMsgFTTag?, 601</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calls getMsgPos2, 602</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calls getMsgPos, 601</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calls getMsgText, 601</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calls poCharPosn, 601</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calls setMsgText, 601</td>
<td></td>
<td></td>
</tr>
<tr>
<td>defun, 601</td>
<td></td>
<td></td>
</tr>
<tr>
<td>putHist, 832</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calledby importFromFrame, 19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calledby recordAndPrint, 282</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calledby restoreHistory, 839</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calledby undoChanges, 835</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calledby undoFromFile, 836</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calledby undoInCore, 834</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calledby writeHistModesAndValues, 845</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calls get, 832</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
INDEX

called by evaluateType1, 1080

called by evaluateType, 1078

called by frameSpad2Cmd, 11

called by funfindLAM, 1060

called by getConstructorDocumentation, 1551

called by getTraceOption, 55

called by hasPair, 1067

called by kcPage, 1515

called by kccPage, 1522

called by kcnPage, 1526

called by kcpPage, 1518

called by mkEvalable, 1075

called by ncAlist, 638

called by ncTag, 637

called by prTraceNames, fn, 80

called by printTypeAndTime, 284

called by processInteractive, 274

called by reportOperations, 1021

called by reportOpsFromUnitDirectly, 1028

called by reportSpadTrace, 76

called by restoreHistory, 839

called by retract, 1203

called by selectOption, 751

called by setExposeAddConstr, 92

called by setExposeAddGroup, 89

called by setExposeAdd, 91

called by setExposeDropConstr, 96

called by setExposeDropGroup, 94

called by setExposeDrop, 94

called by setExpose, 91

called by setInputLibrary, 906

called by setOutputAlgebra, 967

called by setOutputCharacters, 971

called by setOutputFormula, 995

called by setOutputFortran, 974

called by setOutputHtml, 980

called by setOutputMathml, 986

called by setOutputOpenMath, 990

called by setOutputTex, 1002

called by showSpad2Cmd, 1019

called by spadClosure?, 853

called by spadReply, printName, 1069

called by trace1, 48

called by traceReply, 81

called by traceSpad2Cmd, 47

called by transOnlyOption, 1053

called by undoSteps, 38

called by undo, 36

called by untraceDomainConstructor, keepTraced?, 73

called by whatSpad2Cmd, fixpat, 1091

called by writify, writifyInner, 849

called by dbSpecialExpandIfNecessary, 1557

called by getConstructorDocumentation, 1551

called by ?t, 84

called by ScanOrPairVec, ScanOrInner, 858

called by abbreviationsSpad2Cmd, 755

called by countCache, 912

called by dbGetDocTable, ln, 1541

called by dbGetDocTable, 1543

called by dbSpecialExpandIfNecessary, 1557

called by dewritify, dewritifyInner, 854

called by displayProperties, 733

called by domainDescendantsOf, 1507

called by evaluateType1, 1080

called by evaluateType, 1078

called by frameSpad2Cmd, 11

called by getTraceOption, 55

called by hasPair, 1067

called by mkEvalable, 1075

called by ncAlist, 638

called by prTraceNames, fn, 80

called by readHiFi, 843

called by restoreHistory, 839

called by selectOption, 751

called by setExposeAdd, 91

called by setExposeDrop, 94

called by setExpose, 91

called by setInputLibrary, 906

called by setOutputAlgebra, 967

called by setOutputCharacters, 971

called by setOutputFormula, 995

called by setOutputFortran, 974

called by setOutputHtml, 980

called by setOutputMathml, 986

called by setOutputOpenMath, 990

called by setOutputTex, 1002

called by showSpad2Cmd, 1019

called by spadClosure?, 853

called by spadReply, printName, 1069

called by trace1, 48

called by traceReply, 81

called by traceSpad2Cmd, 47
INDEX

1807

called by trace1, 48
called by traceSpad2Cmd, 47
called by transOnlyOption, 1053
called by undoSteps, 38
called by undo, 36
called by writify,writifyInner, 849

called by isSharpVarWithNum, 1063
qenum, 1236
defun, 1236
qlesseqp
called by dbShowCons1, 1547
qrplaca
called by dewritify,dewritifyInner, 854
called by writify,writifyInner, 849
qrplacd
called by dewritify,dewritifyInner, 854
called by writify,writifyInner, 849

qsabsval, 1245
defmacro, 1245
qsadd1, 1244
defmacro, 1244
qsdifference, 1243
defmacro, 1243
qsDot26432, 1256
defmacro, 1256
qsDot2Mod6432, 1256
defmacro, 1256
qsetvelt
called by dewritify,dewritifyInner, 854
called by writify,writifyInner, 849

qslessp, 1244
called by trace1, 48
defmacro, 1244
qsmax, 1246
defmacro, 1246
qsmmin, 1246
defmacro, 1246
qsminus, 1244
defmacro, 1244
qsMod6432, 1255
defmacro, 1255
qsMul6432, 1256
defmacro, 1256
qsMulAdd6432, 1255
defmacro, 1255
qsMulAddMod6432, 1255
defmacro, 1255
qsMulMod32, 1255
defmacro, 1255
qsoddp, 1245
defmacro, 1245
qsplus, 1245
defmacro, 1245
qsquotient, 1243
defun, 1243
qremainder, 1243
defun, 1243
qsub1, 1244
defmacro, 1244
qtimes, 1245
defmacro, 1245
qszerop, 1246
defmacro, 1246
queryClients, 777
called by close, 777
calls sockGetInt, 777
calls sockSendInt, 777
uses $QueryClients, 777
uses $SessionManager, 777
defun, 777
queryUserKeyedMsg
called by close, 777
called by historySpad2Cmd, 822
called by importFromFrame, 19
called by listConstructorAbbreviations, 756
called by quitSpad2Cmd, 872
called by yesanswer, 798
queueUpErrors, 594
called by processMsgList, 591
calls processChPosesForOneLine, 594
uses $outputList, 594
defun, 594
quickAnd
called by domainDescendantsOf, 1507
quit, 871, 872
calls quitSpad2Cmd, 872
defun, 872
manpage, 871
quitSpad2Cmd, 872
called by pquitSpad2Cmd, 870
called by quit, 872
calls leaveScratchpad, 872
calls queryUserKeyedMsg, 872
calls sayKeyedMsg, 872
calls string2id-n, 872
calls tersyscommand, 872
calls upcase, 872
uses $quitCommandType, 872
defun, 872
quoteString, 1449
defun, 1449
quotient
called by Else?, 305
called by ElseIf?, 304
called by If?, 304
called by Top?, 304
called by ptimers, 1052
quotient2, 1239
defun, 1239
qvelt
called by dewritify,dewritifyInner, 854
called by writify,writifyInner, 849
qvlenU16, 1248
defmacro, 1248
qvlenU32, 1249
defmacro, 1249
qvlenU8, 1247
defmacro, 1247
qvmaxindex
called by dewritify,dewritifyInner, 854
called by writify,writifyInner, 849
random, 1239
defun, 1239
rassoc
called by rassocSub, 1058
called by saveMapSig, 53
rassocSub, 1058
called by ?, 84
called by isSubForRedundantMapName, 1059
called by traceReply, 81
calls rassoc, 1058
defun, 1058
rdefinstream, 1240
calls rdefiofstream, 1240
defun, 1240
rdefiofstream
called by rdefinstream, 1240
called by rdefoutstream, 1240
called by readHiFi, 843
called by saveHistory, 837
called by setHistoryCore, 826
called by writeHiFi, 844
rdefoutstream, 1240
calls rdefiofstream, 1240
defun, 1240
rdigit?, 356
calls strpos, 356
defun, 356
read, 874, 875
called by undo, 36
calls readSpad2Cmd, 875
defun, 875
manpage, 874
read-line
called by serverReadLine, 269
readHiFi, 843
called by fetchOutput, 842
called by restoreHistory, 839
called by setHistoryCore, 826
called by showInOut, 842
called by showInput, 841
called by undoFromFile, 836
called by undoInCore, 834
called by writeInputLines, 828
calls assoc, 843
calls histFileName, 843
calls keyedSystemError, 843
calls object2Identifier, 843
calls qcdr, 843
calls readSpad2Cmd, 875
calls rshut, 843
calls spadrread, 843
calls readLibPathFast
called by getDependentsOfConstructor, 1524
called by getUsersOfConstructor, 1526
readline, 1565
defun, 1565
readSpad2Cmd, 875
called by read, 875
calls /read, 875
calls findfile, 875
calls makePathname, 875
calls member, 875
calls mergePathnames, 875
calls namestring, 875
calls optionError, 875
calls pathnameName, 875
calls pathnameTypeld, 875
calls pathname, 875
calls selectOptionLC, 875
calls throwKeyedMsg, 875
calls upcase, 875
uses /editfile, 875
uses $InteractiveMode, 875
uses $UserLevel, 875
defun, 875
readSpadProfileIfThere, 1107
called by restart, 239
uses /editfile, 1107
defun, 1107
reassembleTowerIntoType, 689
called by computeTTTranspositions, 686
called by reassembleTowerIntoType, 689
defun, 689
recover, 263
called by clearCmdCompletely, 770
defun, 263
recordAndPrint, 282
called by processInteractive1, 277
calls mkCompanionPage, 282
calls mkObjWrap, 282
calls output, 282
calls printStatisticsSummary, 282
calls printStorage, 282
calls printTypeAndTime, 282
calls putHist, 282
calls recordAndPrintTest, 282
uses $collectOutput, 282
uses $e, 282
uses $mkTestFlag, 282
uses $mkTestOutputType, 282
uses $outputMode, 282
uses $printAnyIfTrue, 282
uses $printStatisticsSummaryIfTrue, 282
uses $printStorageIfTrue, 282
uses $printTimeIfTrue, 282
uses $printVoidIfTrue, 282
uses $runTestFlag, 282
defun, 282
recordAndPrintTest
called by recordAndPrint, 282
recordFrame, 1083
called by processInteractive1, 277
called by undoSteps, 38
called by diffAlist, 1083
called by exit, 1083
called by seq, 1083
called by putHist, 1083
uses $InteractiveFrame, 1083
uses $frameRecord, 1083
uses $previousBindings, 1083
defun, 1083
recordNewValue, 833
called by clearCmdParts, 773
called by putHist, 832
called by undoFromFile, 836
called by recordNewValue0, 833
called by startTimingProcess, 833
called by stopTimingProcess, 833
defun, 833
recordNewValue0, 833
called by recordNewValue, 833
called by assq, 833
uses $HistRecord, 833
defun, 833
recordOldValue, 833
called by clearCmdParts, 773
called by putHist, 832
called by undoFromFile, 836
called by recordOldValue0, 833
called by startTimingProcess, 833
called by stopTimingProcess, 833
defun, 833
recordOldValue0, 834
  called by recordOldValue, 833
  calls assq, 833
  uses $HistList, 834
defun, 834
reduce-stack-clear
called by ioclear, 1118
reduceAlistForDomain, 1519
called by kccPage, 1522
calls nreverse0, 1520
calls simpHasPred, 1520
calls sublislis, 1519
defun, 1519
redundant, 596
calls msgNoRep?, 596
calls sameMsg?, 596
uses $noRepList, 596
defun, 596
refvecp
called by isDomainOrPackage, 1061
called by spadTrace, 67
regress, 878, 882
calls findnexttest, 882
calls getspoolname, 882
calls testpassed, 882
uses *all-tests-run*, 882
defun, 882
manpage, 878
remainder
called by KeepPart?, 305
called by SkipEnd?, 305
called by SkipPart?, 305
remainder2, 1239
defun, 1239
remalist
called by clearParserMacro, 725
rendup
called by clearCmdParts, 773
called by computeTTTranspositions, 686
called by dbConsHeading, 1553
called by dbShowCons1, 1547
called by dbShowConsDoc, 1549
called by displayMacros, 799
called by displayOperationsFromLisplib, 1027
called by displayProperties, 733
called by reportOpsFromLisplib, 1025
called by reportOpsFromUnitDirectly, 1028
called by retract2Specialization, 704
remFile, 579
called by getPosStL, 578
calls ifcdr, 579
defun, 579
remLine, 584
called by getPosStL, 578
calls ifcar, 579
defun, 584
removeAttributes
called by getMsgInfoFromKey, 588
removeOption, 1054
called by spadTrace, 67
defun, 1054
removeQuote
called by coerceBranch2Union, 698
remover, 1070
called by remover, 1070
called by spadUntrace, 78
calls remover, 1070
defun, 1070
removeTracedMapSigs, 61
called by untrace, 60
uses $tracedMapSignatures, 61
defun, 61
removeUndoLines, 41
calls charPosition, 41
calls concat, 41
calls exit, 41
calls maxindex, 41
calls seq, 41
calls stringPrefix?, 41
calls substring, 41
calls trimString, 41
calls undoCount, 41
uses $IOindex, 41
uses $currentLine, 41
defun, 41
removeZeroOne
called by getSubDomainPredicate, 701
removeZeroOneDestructively
called by reportOperations, 1021
remprop
called by loadLib, 1182
<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page/Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>rempropI</td>
<td>839</td>
</tr>
<tr>
<td>called by restoreHistory</td>
<td></td>
</tr>
<tr>
<td>renamePatternVariables</td>
<td>1431</td>
</tr>
<tr>
<td>defun</td>
<td></td>
</tr>
<tr>
<td>renamePatternVariables1</td>
<td>1431</td>
</tr>
<tr>
<td>defun</td>
<td></td>
</tr>
<tr>
<td>rep</td>
<td>593</td>
</tr>
<tr>
<td>called by makeMsgFromLine</td>
<td>593</td>
</tr>
<tr>
<td>defun</td>
<td>1021</td>
</tr>
<tr>
<td>replaceLast</td>
<td>682</td>
</tr>
<tr>
<td>called by coerceIntTower</td>
<td></td>
</tr>
<tr>
<td>replacePercentByDollar</td>
<td>1410</td>
</tr>
<tr>
<td>defun</td>
<td>1409</td>
</tr>
<tr>
<td>replacePercentByDollar.fn</td>
<td>1409</td>
</tr>
<tr>
<td>defun</td>
<td></td>
</tr>
<tr>
<td>replaceSharps</td>
<td>1104</td>
</tr>
<tr>
<td>called by evaluateType1</td>
<td>1080</td>
</tr>
<tr>
<td>called by valueArgsEqual?</td>
<td>696</td>
</tr>
<tr>
<td>calls subCopy</td>
<td>1104</td>
</tr>
<tr>
<td>local ref $FormalMapVariableList</td>
<td>1104</td>
</tr>
<tr>
<td>local ref $TriangleVariableList</td>
<td>1104</td>
</tr>
<tr>
<td>defun</td>
<td>1021</td>
</tr>
<tr>
<td>reportInstantiations</td>
<td>274</td>
</tr>
<tr>
<td>called by processInteractive</td>
<td></td>
</tr>
<tr>
<td>reportInstantiations,1205</td>
<td>1021</td>
</tr>
<tr>
<td>defun</td>
<td>1021</td>
</tr>
<tr>
<td>reportOperations</td>
<td>1021</td>
</tr>
<tr>
<td>called by showSpad2Cmd</td>
<td>1019</td>
</tr>
<tr>
<td>calls bright</td>
<td>1021</td>
</tr>
<tr>
<td>calls evaluateType</td>
<td>1021</td>
</tr>
<tr>
<td>calls isDomainValuedVariable</td>
<td>1021</td>
</tr>
<tr>
<td>calls isNameOfType</td>
<td>1021</td>
</tr>
<tr>
<td>calls isType</td>
<td>1021</td>
</tr>
<tr>
<td>calls mkAtree</td>
<td>1021</td>
</tr>
<tr>
<td>calls opOf</td>
<td>1021</td>
</tr>
<tr>
<td>calls qcar</td>
<td>1021</td>
</tr>
<tr>
<td>calls removeZeroOneDestructively</td>
<td>1021</td>
</tr>
<tr>
<td>calls reportOpsFromLispLib0</td>
<td>1021</td>
</tr>
<tr>
<td>calls reportOpsFromUnitDirectly0</td>
<td>1021</td>
</tr>
<tr>
<td>calls sayBrightly</td>
<td>1021</td>
</tr>
<tr>
<td>calls sayKeyedMsg</td>
<td>1021</td>
</tr>
<tr>
<td>calls unabbrev</td>
<td>1021</td>
</tr>
<tr>
<td>uses $doNotAddEmptyModeIfTrue</td>
<td>1021</td>
</tr>
<tr>
<td>uses $env</td>
<td>1022</td>
</tr>
<tr>
<td>uses $eval</td>
<td>1022</td>
</tr>
<tr>
<td>uses $genValue</td>
<td>1022</td>
</tr>
<tr>
<td>uses $quadSymbol</td>
<td>1022</td>
</tr>
<tr>
<td>defun</td>
<td>1021</td>
</tr>
<tr>
<td>reportOpsFromLispLib</td>
<td>1024</td>
</tr>
<tr>
<td>called by reportOpsFromLispLib0</td>
<td>1023</td>
</tr>
<tr>
<td>called by reportOpsFromLispLib1</td>
<td>1024</td>
</tr>
<tr>
<td>calls bright</td>
<td>1024</td>
</tr>
<tr>
<td>calls concat</td>
<td>1024</td>
</tr>
<tr>
<td>calls constructor?</td>
<td>1024</td>
</tr>
<tr>
<td>calls dc1</td>
<td>1025</td>
</tr>
<tr>
<td>calls displayOperationsFromLispLib</td>
<td>1025</td>
</tr>
<tr>
<td>calls eqlsublist</td>
<td>1024</td>
</tr>
<tr>
<td>calls form2StringWithWhere</td>
<td>1024</td>
</tr>
<tr>
<td>calls form2String</td>
<td>1025</td>
</tr>
<tr>
<td>calls formatAttribute</td>
<td>1025</td>
</tr>
<tr>
<td>calls getConstructorSignature</td>
<td>1024</td>
</tr>
<tr>
<td>calls getdatabase</td>
<td>1024</td>
</tr>
<tr>
<td>calls ifcdr</td>
<td>1024</td>
</tr>
<tr>
<td>calls isExposedConstructor</td>
<td>1024</td>
</tr>
<tr>
<td>calls msort</td>
<td>1025</td>
</tr>
<tr>
<td>calls namestring</td>
<td>1024</td>
</tr>
<tr>
<td>calls nreverse0</td>
<td>1024</td>
</tr>
<tr>
<td>calls remdup</td>
<td>1025</td>
</tr>
<tr>
<td>calls say2PerLine</td>
<td>1025</td>
</tr>
<tr>
<td>calls sayBrightly</td>
<td>1024</td>
</tr>
<tr>
<td>calls sayKeyedMsg</td>
<td>1024</td>
</tr>
<tr>
<td>calls selectOptionLC</td>
<td>1025</td>
</tr>
<tr>
<td>calls specialChar</td>
<td>1025</td>
</tr>
<tr>
<td>calls strconc</td>
<td>1024</td>
</tr>
<tr>
<td>uses $FormalMapVariableList</td>
<td>1025</td>
</tr>
<tr>
<td>uses $linelength</td>
<td>1025</td>
</tr>
<tr>
<td>uses $options</td>
<td>1025</td>
</tr>
<tr>
<td>uses $showOptions</td>
<td>1025</td>
</tr>
<tr>
<td>defun</td>
<td>1024</td>
</tr>
<tr>
<td>reportOpsFromLispLib0</td>
<td>1023</td>
</tr>
<tr>
<td>called by reportOperations</td>
<td>1021</td>
</tr>
<tr>
<td>called by reportOpsFromLispLib1</td>
<td>1023</td>
</tr>
<tr>
<td>uses $useEditorForShowOutput</td>
<td>1023</td>
</tr>
<tr>
<td>defun</td>
<td>1023</td>
</tr>
<tr>
<td>reportOpsFromLispLib1</td>
<td>1024</td>
</tr>
<tr>
<td>called by reportOpsFromLispLib0</td>
<td>1023</td>
</tr>
<tr>
<td>called by reportOpsFromLispLib1</td>
<td>1024</td>
</tr>
<tr>
<td>calls defiostream</td>
<td>1024</td>
</tr>
</tbody>
</table>
calls editFile, 1024
calls erase, 1024
calls pathname, 1024
calls reportOpsFromLisplib, 1024
calls sayShowWarning, 1024
calls shut, 1024
uses $erase, 1024
uses $sayBrightlyStream, 1024
defun, 1024

reportOpsFromUnitDirectly, 1028
called by displayOperationsFromLisplib, 1027
called by reportOpsFromUnitDirectly0, 1027
called by reportOpsFromUnitDirectly1, 1032
calls bright, 1028
calls concat, 1028
calls evalDomain, 1028
calls formatAttribute, 1028
calls formatOpType, 1028
calls formatOperation, 1028
calls getOplistForConstructorForm, 1028
calls getdatabase, 1028
calls getl, 1028
calls isExposedConstructor, 1028
calls member, 1028
calls msort, 1028
calls namestring, 1028
calls nreverse0, 1028
calls qcar, 1028
calls remdup, 1028
calls say2PerLine, 1028
calls sayBrightly, 1028
calls systemErrorHere, 1028
uses $CategoryFrame, 1028
uses $commentedOps, 1028
uses $linelength, 1028
uses $options, 1028
uses $showOptions, 1028
defun, 1028

reportOpsFromUnitDirectly0, 1027
called by reportOperations, 1021
calls reportOpsFromUnitDirectly1, 1027
calls reportOpsFromUnitDirectly, 1027
uses $useEditorForShowOutput, 1028
defun, 1027
reportOpsFromUnitDirectly1, 1032
called by reportOpsFromUnitDirectly0, 1027
calls defostream, 1032
calls editFile, 1032
calls erase, 1032
calls pathname, 1032
calls reportOpsFromUnitDirectly, 1032
calls sayShowWarning, 1032
calls shut, 1032
uses $erase, 1032
uses $sayBrightlyStream, 1032
defun, 1032

reportOpSymbol
called by displayOperations, 798
reportSpadTrace, 76
called by ?t, 84
called by spadTrace,isTraceable, 66
called by spadTrace, 67
calls qcar, 76
calls sayBrightly, 76
uses $traceNoisely, 76
defun, 76
reportUndo, 43
called by diffAlist, 1085
calls concat, 44
calls exit, 44
calls lassoc, 44
calls pname, 44
calls pp, 44
calls qcar, 44
calls sayBrightlyNT, 44
calls sayBrightly, 44
calls seq, 43
defun, 43
reportundo
called by $InteractiveFrame, 44
reportWhatOptions, 1093
called by whatSpad2Cmd, 1091
calls sayBrightly, 1093
uses $whatOptions, 1093
defun, 1093
reroot, 264
called by initroot, 259
calls make-absolute-filename, 265
uses $current-directory, 265
uses $directory-list, 265
uses $library-directory-list, 265
uses $relative-directory-list, 265
uses $relative-library-directory-list, 265
uses $spadroot, 265
defun, 264
reset-stack-limits
called by resetStackLimits, 244
resetCounters, 1051
called by trace1, 48
calls concat, 1051
uses /countlist, 1051
defun, 1051
resetHashTables, 1146
calls browseopen, 1146
calls categoryopen, 1146
calls initial-getdatabase, 1146
calls interpopen, 1146
calls operationopen, 1146
uses *allconstructors*, 1147
uses *browse-stream*, 1146
uses *category-stream*, 1146
uses *category-stream-stamp*, 1146
uses *hascategory-hash*, 1147
uses *interp-stream*, 1146
uses *interp-stream-stamp*, 1147
uses *operation-hash*, 1147
uses *operation-stream*, 1146
uses *operation-stream-stamp*, 1146
uses *sourcefiles*, 1146
defun, 1146
resetInCoreHist, 830
called by clearCmdAll, 771
called by historySpad2Cmd, 822
uses $HistListAct, 830
uses $HistListLen, 830
uses $HistList, 830
defun, 830
resetSpacers, 1051
called by trace1, 48
calls concat, 1051
uses /spacelist, 1051
defun, 1051
resetStackLimits, 244
called by intloop, 250
called by protectedEVAL, 271
called by runspad, 243
calls reset-stack-limits, 244
defun, 244
resetTimers, 1051
called by trace1, 48
calls concat, 1051
uses /timerlist, 1051
defun, 1051
resetWorkspaceVariables, 894
calls copy, 894
calls initializeSetVariables, 894
uses /countlist, 894
uses /editFile, 894
uses /pretty, 894
uses /sourcefiles, 894
uses /spacelist, 894
uses /timerlist, 894
uses $CommandSynonymAlist, 895
uses $IOindex, 895
uses $InitialCommandSynonymAlist, 895
uses $UserAbbreviationsAlist, 895
uses $boot, 894
uses $coerceIntByMapCounter, 895
uses $compileMapFlag, 894
uses $dependeeClosureAlist, 895
uses $echoLineStack, 895
uses $env, 895
uses $existingFiles, 894
uses $e, 895
uses $functionTable, 894
uses $msgAlist, 895
uses $msgDatabaseName, 895
uses $msgDatabase, 895
uses $operationNameList, 895
uses $setOptions, 895
uses $slamFlag, 895
uses $sourceFiles, 894
defun, 894
resolveTT
called by unifyStructVar, 659
resolveTypeListAny
called by retract2Specialization, 704
Rest, 302
called by include1, 306
defmacro, 302
restart
calls get-current-directory, 239
calls init-memory-config, 239
calls initHist, 239
calls initializeInterpreterFrameRing, 239
calls initroot, 239
calls makeInitialModemapFrame, 239
calls openserver, 239
calls readSpadProfileIfThere, 239
calls restart0, 239
calls spadStartUpMsgs, 239
calls spad, 239
calls statisticsInitialization, 239
uses $IOindex, 239
uses $InteractiveFrame, 239
uses $SpadServerName, 239
uses $SpadServer, 239
uses $current-directory, 239
uses $currentLine, 239
uses $displayStartMsgs, 239
uses $openServerIfTrue, 239
uses $printLoadMsgs, 239
restart function, 238
restart0, 241
calledby restart, 239
calls browseopen, 241
calls categoryopen, 241
calls interpopen, 241
calls operationopen, 241
defun, 241
restoreHistory, 839
calledby historySpad2Cmd, 822
calls $fcopy, 839
calls clearCmdSortedCaches, 839
calls clearSpad2Cmd, 839
calls disableHist, 839
calls get, 839
calls histFileErase, 839
calls histFileName, 839
calls identp, 839
calls makeHistFileName, 839
calls makeInputFilename, 839
calls namestring, 839
calls putHist, 839
calls qcar, 839
calls qcdr, 839
calls readHiFi, 839
calls rempropI, 839
calls rkeyids, 839
calls sayKeyedMsg, 839
calls throwKeyedMsg, 839
calls updateInCoreHist, 839
uses $HiFiAccess, 839
uses $InteractiveFrame, 839
uses $e, 839
uses $internalHistoryTable, 839
uses $oldHistoryFileName, 839
uses $options, 839
uses $useInternalHistoryTable, 839
defun, 839
resultp, 888
calledby split, 886
defun, 888
retract, 1203
calledby coerceOrRetract, 703
calledby printTypeAndTime, 284
calledby retract2Specialization, 704
calls isWrapped, 1203
calls mkObj, 1203
calls objMode, 1203
calls objVal, 1203
calls qcar, 1203
calls retract1, 1203
local ref $EmptyMode, 1203
defun, 1203
retract1
calledby retract, 1203
retract2Specialization, 704
calls coerceInt, 704
calls coerceUnion2Branch, 704
calls get, 704
calls isPartialMode, 704
calls isRectangularList, 704
calls member, 704
calls mkObjWrap, 704
calls mkObj, 704
calls objMode, 704
calls objValUnwrap, 704
calls objVal, 704
calls remdup, 704
calls resolveTypeListAny, 704
calls retract, 704
calls unwrap, 704
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>calls varsInPoly</td>
<td>704</td>
</tr>
<tr>
<td>uses $Any</td>
<td>704</td>
</tr>
<tr>
<td>uses $Integer</td>
<td>704</td>
</tr>
<tr>
<td>uses $NonNegativeInteger</td>
<td>704</td>
</tr>
<tr>
<td>uses $PositiveInteger</td>
<td>704</td>
</tr>
<tr>
<td>uses $QuotientField</td>
<td>704</td>
</tr>
<tr>
<td>uses $Symbol</td>
<td>704</td>
</tr>
<tr>
<td>uses $e</td>
<td>704</td>
</tr>
<tr>
<td>defun</td>
<td>704</td>
</tr>
<tr>
<td>retractByFunction</td>
<td>710</td>
</tr>
<tr>
<td>called by coerceRetract</td>
<td>709</td>
</tr>
<tr>
<td>calls coerceUnion2Branch</td>
<td>710</td>
</tr>
<tr>
<td>calls compiledLookup</td>
<td>710</td>
</tr>
<tr>
<td>calls evalDomain</td>
<td>710</td>
</tr>
<tr>
<td>calls findFunctionInDomain</td>
<td>710</td>
</tr>
<tr>
<td>calls mkObjWrap</td>
<td>710</td>
</tr>
<tr>
<td>calls objMode</td>
<td>710</td>
</tr>
<tr>
<td>calls objValUnwrap</td>
<td>710</td>
</tr>
<tr>
<td>calls orderMms</td>
<td>710</td>
</tr>
<tr>
<td>calls sayFunctionResult</td>
<td>710</td>
</tr>
<tr>
<td>calls sayFunctionSelection</td>
<td>710</td>
</tr>
<tr>
<td>calls spadcall</td>
<td>710</td>
</tr>
<tr>
<td>uses $dollar</td>
<td>710</td>
</tr>
<tr>
<td>uses $reportBottomUpFlag</td>
<td>710</td>
</tr>
<tr>
<td>defun</td>
<td>710</td>
</tr>
<tr>
<td>retractUnderDomain</td>
<td>709</td>
</tr>
<tr>
<td>calls coerceInt</td>
<td>709</td>
</tr>
<tr>
<td>calls constructT</td>
<td>709</td>
</tr>
<tr>
<td>calls deconstructT</td>
<td>709</td>
</tr>
<tr>
<td>calls nequal</td>
<td>709</td>
</tr>
<tr>
<td>calls underDomainOf</td>
<td>709</td>
</tr>
<tr>
<td>defun</td>
<td>709</td>
</tr>
<tr>
<td>rkeyids</td>
<td>709</td>
</tr>
<tr>
<td>called by restoreHistory</td>
<td>839</td>
</tr>
<tr>
<td>called by setHistoryCore</td>
<td>826</td>
</tr>
<tr>
<td>rshut</td>
<td></td>
</tr>
<tr>
<td>called by getDependentsOfConstructor</td>
<td>1525</td>
</tr>
<tr>
<td>called by getUsersOfConstructor</td>
<td>1526</td>
</tr>
<tr>
<td>called by readHiFi</td>
<td>843</td>
</tr>
<tr>
<td>called by saveDependentsHashTable</td>
<td>1168</td>
</tr>
<tr>
<td>called by saveHistory</td>
<td>837</td>
</tr>
<tr>
<td>called by saveUsersHashTable</td>
<td>1169</td>
</tr>
<tr>
<td>called by setHistoryCore</td>
<td>826</td>
</tr>
<tr>
<td>called by writeHiFi</td>
<td>844</td>
</tr>
<tr>
<td>ruleLhsTran</td>
<td>552</td>
</tr>
<tr>
<td>called by pfRule2Sex</td>
<td>548</td>
</tr>
<tr>
<td>called by nsubst</td>
<td>552</td>
</tr>
<tr>
<td>called by patternVarsOf</td>
<td>552</td>
</tr>
<tr>
<td>uses $multiVarPredicateList</td>
<td>552</td>
</tr>
<tr>
<td>uses $predicateList</td>
<td>552</td>
</tr>
<tr>
<td>defun</td>
<td>552</td>
</tr>
<tr>
<td>rulePredicateTran</td>
<td>549</td>
</tr>
<tr>
<td>called by pfRule2Sex</td>
<td>548</td>
</tr>
<tr>
<td>called by patternVarsOf</td>
<td>550</td>
</tr>
<tr>
<td>called by pvarPredTran</td>
<td>550</td>
</tr>
<tr>
<td>uses $multiVarPredicateList</td>
<td>550</td>
</tr>
<tr>
<td>defun</td>
<td>549</td>
</tr>
<tr>
<td>runspad</td>
<td>243</td>
</tr>
<tr>
<td>called by spad</td>
<td>242</td>
</tr>
<tr>
<td>called by exit</td>
<td>243</td>
</tr>
<tr>
<td>called by ncTopLevel</td>
<td>243</td>
</tr>
<tr>
<td>called by resetStackLimits</td>
<td>243</td>
</tr>
<tr>
<td>called by seq</td>
<td>243</td>
</tr>
<tr>
<td>called by $quitTag</td>
<td>243</td>
</tr>
<tr>
<td>catches</td>
<td>243</td>
</tr>
<tr>
<td>defun</td>
<td>243</td>
</tr>
<tr>
<td>rwrite</td>
<td>846</td>
</tr>
<tr>
<td>called by rwrite</td>
<td>846</td>
</tr>
<tr>
<td>called by saveDependentsHashTable</td>
<td>1168</td>
</tr>
<tr>
<td>called by saveUsersHashTable</td>
<td>1168</td>
</tr>
<tr>
<td>called by spadrwrite0</td>
<td>846</td>
</tr>
<tr>
<td>calls identp</td>
<td>847</td>
</tr>
<tr>
<td>calls $name</td>
<td>846</td>
</tr>
<tr>
<td>calls rwrite</td>
<td>846</td>
</tr>
<tr>
<td>defun</td>
<td>846</td>
</tr>
<tr>
<td>rread</td>
<td>847</td>
</tr>
<tr>
<td>called by SPADRREAD</td>
<td>847</td>
</tr>
<tr>
<td>called by getDependentsOfConstructor</td>
<td>1526</td>
</tr>
<tr>
<td>called by getUsersOfConstructor</td>
<td>1526</td>
</tr>
<tr>
<td>called by rread</td>
<td>847</td>
</tr>
</tbody>
</table>

rwrite, 846
   called by rwrite, 846
called by saveDependentsHashTable, 1168
called by saveUsersHashTable, 1168
called by spadrwrite0, 846
calls identp, 847, 848
calls $name, 846, 848
calls rwrite, 846
defun, 846

rread, 847
   called by SPADRREAD, 847
called by getDependentsOfConstructor, 1526
called by getUsersOfConstructor, 1526
called by rread, 847
catches, 849
defun, 849
sameMsg?, 598
called by redundant, 596
calls getMsgArgL, 598
calls getMsgKey, 598
defun, 598
sameUnionBranch, 285
called by printTypeAndTime, 284
defun, 285
satBreak
called by kcPage, 1516
satisfiesRegularExpressions, 1095
called by filterListOfStringsWithFn, 1095
called by filterListOfStrings, 1094
calls strpos, 1095
defun, 1095
satisfiesUserLevel, 723
called by commandsForUserLevel, 720
called by displaySetVariableSettings, 897
called by set1, 1014
uses $UserLevel, 723
defun, 723
save-system
called by spad-save, 1136
saveDependentsHashTable, 1168
called by make-databases, 1165
calls erase, 1168
calls hget, 1168
calls hkeys, 1168
calls msort, 1168
calls rshut, 1168
calls rwrite, 1168
calls writeLib1, 1168
local ref $DepTb, 1168
local ref $erase, 1168
defun, 1168
saveHistory, 837
called by historySpad2Cmd, 822
calls histFileErase, 837
calls histFileName, 837
calls histInputFileName, 837
calls makeHistFileName, 837
calls namestring, 837
calls object2Identifier, 837
calls redefineStream, 837
calls rshut, 837
calls sayKeyedMsg, 837
calls spadrwrite0, 837
calls throwKeyedMsg, 837
calls writeInputLines, 837
uses $HiFiAccess, 837
uses $internalHistoryTable, 837
uses $seen, 837
uses $useInternalHistoryTable, 837
defun, 837
saveMapSig, 53
called by trace1, 49
calls addassoc, 53
calls getMapSig, 53
calls rassoc, 53
uses $mapSubNameAlist, 53
uses $tracedMapSignatures, 53
defun, 53
saveSystem, 890, 891
calls helpSpad2Cmd, 891
calls spad-save, 891
defun, 891
manpage, 890
saveUsersHashTable, 1168
called by make-databases, 1165
calls erase, 1168
calls hget, 1169
calls hkeys, 1168
calls msort, 1168
calls rshut, 1169
calls rwrite, 1168
calls writeLib1, 1168
local ref $erase, 1169
local ref $usersTb, 1169
defun, 1168
say
called by displaySetVariableSettings, 897
called by newHelpSpad2Cmd, 813
called by trace1, 48
called by whatCommands, 1093
called by workFilesSpad2Cmd, 1099
say2PerLine
called by displayOperationsFromLisplib, 1027
called by reportOpsFromLisplib, 1025
sayFunctionSelectionResult called by coerceIntByMap, 693
called by retractByFunction, 710
sayKeyedMsg, 27
called by abbreviationsSpad2Cmd, 755
called by apropos, 1098
called by changeHistListLen, 830
called by clearCmdAll, 771
called by clearCmdCompletely, 770
called by clearCmdParts, 773
called by clearSpad2Cmd, 767
called by commandAmbiguityError, 724
called by commandErrorMessage, 722
called by countCache, 912
called by describeSetStreamsCalculate, 1007
called by displayExposedConstructors, 97
called by displayExposedGroups, 97
called by displayHiddenConstructors, 98
called by displayOperations, 798
called by displayProperties, 733
called by handleNoParseCommands, 742
called by helpSpad2Cmd, 813
called by historySpad2Cmd, 822
called by history, 821
called by importFromFrame, 19
called by listConstructorAbbreviations, 756
called by loadLibNoUpdate, 1183
called by loadLib, 1181
called by localdatabase, 1160
called by localnrlib, 1161
called by newHelpSpad2Cmd, 813
called by npsystem, 744
called by printStatisticsSummary, 283
called by printTypeAndTime, 284
called by quitSpad2Cmd, 872
called by reportOperations, 1021
called by reportOpsFromLisplib, 1024
called by restoreHistory, 839
called by saveHistory, 837
called by set1, 1014
called by setExposeAddConstr, 92
called by setExposeAddGroup, 89
called by setExposeAdd, 91
called by setExposeDropConstr, 96
called by setExposeDropGroup, 95
called by setExposeDrop, 94
called by setExpose, 91
called by setHistoryCore, 826
called by setOutputAlgebra, 967
called by setOutputFormula, 995
called by setOutputFortran, 974
called by setOutputHtml, 980
called by setOutputMathml, 986
called by setOutputOpenMath, 991
called by setOutputTex, 1002
called by showHistory, 824
called by showSpad2Cmd, 1019
called by spadStartUpMsgs, 241
called by trace1, 48
called by undoInCore, 834
called by leveledError, 723
called by whatCommands, 1094
called by whatSpad2Cmd, 1091
called by workfilesSpad2Cmd, 1099
called by writeInputLines, 828
called by writeComplain, 848
calls sayKeyedMsgLocal, 27
defun, 27
sayKeyedMsgLocal, 28
called by sayKeyedMsg, 27
calls flowSegmentedMsg, 28
calls sayMSG2File, 28
calls sayMSG, 28
calls segmentKeyedMsg, 28
calls substituteSegmentedMsg, 28
calls $displayMsgNumber, 28
calls $linelength, 28
calls $margin, 28
calls $printMsgsToFile, 28
defun, 28
sayMessage
called by apropos, 1097
called by clearCmdParts, 773
called by describeSpad2Cmd, 790
called by displaySetOptionInformation, 896
called by displaySpad2Cmd, 796
called by displayWorkspaceNames, 726
called by filterAndFormatConstructors, 1096
called by printLabeledList, 746
called by set1, 1015
called by setFortPers, 961
INDEX

sayMSG, 29
  called by setFunctionsCache, 911
  called by setOutputCharacters, 971
  called by setStreamsCalculate, 1007
  called by traceReply, 80
  called by updateFromCurrentInterpreterFrame, 15
  called by whatCommands, 1093

sayNewLine
  called by sayexample, 801

sayShowWarning, 1032
  called by reportOpsFromLisplib1, 1024
  called by reportOpsFromUnitDirectly1, 1032
  calls sayBrightly, 1032
defun, 1032

scanCheckRadix, 357
  called by scanNumber, 355
  uses $linepos, 357
defun, 357

scanCloser, 348
  used by scanCloser?, 349
defvar, 348

scanCloser?, 349
called by scanKeyTr, 343
calls keyword, 349
defun, 349

scanComment, 339
called by scanToken, 337
calls lfcomment, 339
calls substring, 339
defun, 339
defvar, 339

scanDict, 360
called by scanDictCons, 360
calls hkeys, 360
defun, 360
defvar, 360

scanError, 358
called by scanPunct, 341
called by scanToken, 337
calls lferror, 358
calls lnExtraBlanks, 358
calls ncSoftError, 358
defun, 358
defvar, 358
defus, 358

saymsg
calls saybrightly1, 29

sayMSG2File, 28
called by sayKeyedMsgLocal, 28
calls defiostream, 28
calls makePathname, 28
calls sayBrightly1, 28
calls shut, 28
defun, 28
called by scanS, 354
called by scanW, 351
called by spleI, 346
calls nextline, 347
calls scanEsc, 347
calls startsComment?, 347
calls startsNegComment?, 347
calls strposl, 347
uses $ln, 347
uses $$n, 347
uses $$r, 347
uses $$sz, 347
defun, 346
scanEscape, 358
called by scanToken, 337
calls scanEsc, 358
calls scanWord, 358
uses $n, 358
defun, 358
scanExponent, 349
called by scanNumber, 355
called by scanPossFloat, 344
calls concat, 350
calls digit?, 350
calls lffloat, 349
calls spleI, 350
uses $ln, 350
uses $$n, 350
uses $$sz, 350
defun, 349
scanIgnoreLine, 336
called by lineoftoks, 335
calls incPrefix?, 336
defun, 336
scanInsert, 361
defun, 361
scanKeyTable, 359
defvar, 359
scanKeyTableCons, 359
defun, 359
scanKeyTr, 343
called by scanPunct, 341
calls keyword, 343
calls lfkey, 343
calls scanCloser?, 343
calls scanPossFloat, 343
uses $floatok, 343
defun, 343
scanKeyWords, 331
defvar, 331
scanNegComment, 340
called by scanToken, 337
calls lfnegcomment, 340
calls substring, 340
uses $ln, 340
uses $$n, 340
uses $$sz, 340
defun, 340
scanNumber, 355
called by scanToken, 337
calls concat, 355
calls lffloat, 355
calls lffrinteger, 355
calls scanCheckRadix, 355
calls scanExponent, 355
calls spleI, 355
calls spleI1, 355
uses $ln, 355
uses $$n, 355
uses $$sz, 355
defun, 355
ScanOrPairVec, 858
called by dewritify, 857
called by writify, 852
calls ScanOrPairVec, ScanOrInner, 858
uses $seen, 858
catches, 858
defun, 858
ScanOrPairVec, ScanOrInner, 857
called by ScanOrPairVec, ScanOrInner, 858
called by ScanOrPairVec, 858
calls ScanOrPairVec, ScanOrInner, 858
calls hget, 858
calls hput, 858
calls qcar, 858
calls qcdr, 858
calls vecp, 858
uses $seen, 858
defun, 857
throws, 857
scanPossFloat, 344
called by scanKeyTr, 343
calls lfkey, 344
calls scanExponent, 344
calls spleI, 344
uses $ln, 344
uses $n, 344
uses $sz, 344
defun, 344
scanPun, 362
defvar, 362
scanPunCons, 362
calls lkeys, 363
defun, 362
scanPunct, 341
called by scanToken, 337
calls scanError, 341
calls scanKeyTr, 341
calls subMatch, 341
uses $ln, 341
uses $n, 341
defun, 341
scanS, 354
called by scanString, 353
called by scanS, 354
calls concat, 354
calls lnExtraBlanks, 354
calls ncSoftError, 354
calls scanEsc, 354
calls scanS, 354
calls scanS, 354

calls scanS, 354
calls substring, 354
uses $linepos, 354
uses $ln, 354
uses $n, 354
uses $sz, 354
defun, 354
scanSpace, 352
called by scanToken, 337
calls lfspaces, 352
calls strposl, 352
uses $floatok, 352
uses $ln, 352
uses $n, 352
defun, 352
scanString, 353
called by scanToken, 337
calls lfstring, 353
calls scanS, 353
uses $floatok, 353
uses $n, 353
defun, 353
scanToken, 337
calls constoken, 338
calls dqUnit, 338
calls lfd, 337
calls lnExtraBlanks, 338
calls punctuation?, 337
calls scanComment, 337
calls scanError, 337
calls scanEscape, 337
calls scanNegComment, 337
calls scanNumber, 337
calls scanPunct, 337
calls scanSpace, 337
calls scanString, 337
calls scanWord, 337
calls startsComment?, 337
calls startsId?, 337
calls startsNegComment?, 337
uses $linepos, 338
uses $ln, 338
uses $n, 338
defun, 337
scanTransform, 355
called by scanS, 354
defun, 355
scanW, 351
called by scanWord, 349
called by scanW, 351
calls concat, 351
calls idChar?, 351
calls posend, 351
calls scanEsc, 351
calls scanW, 351
calls substring, 351
uses $ln, 351
uses $n, 351
uses $sz, 351
defun, 351
scanWord, 349
called by scanEscape, 358
called by scanToken, 337

calls keyword?, 349
calls lfd, 349
calls lfkey, 349
calls scanW, 349
uses $floatok, 349
defun, 349

search, 1110
called by getProplist, 1110
calls searchCurrentEnv, 1110
calls searchTailEnv, 1110
defun, 1110

searchCurrentEnv, 1110
called by search, 1110
calls assq, 1110
defun, 1110

searchTailEnv, 1111
called by search, 1110
calls assq, 1111
defun, 1111

sec, 1221
defun, 1221

sech, 1223
defun, 1223

segmentKeyedMsg, 28
called by getMsgInfoFromKey, 588
called by msgText, 286
called by sayKeyedMsgLocal, 28
calls string2Words, 28
defun, 28

selectLocalMms
called by coerceInt1, 676
called by getSubDomainPredicate, 701

selectMmsI
called by coerceConvertMmSelection, 684
called by coerceInt1, 676
called by coerceIntByMap, 693

selectOption, 751
called by selectOptionLC, 751
called by set1, 1014
called by systemCommand, 720
called by unAbbreviateKeyword, 741
calls identp, 751
calls member, 751
calls pname, 751
calls qcar, 751

calls qcdr, 751
calls stringPrefix?, 751
defun, 751

selectOptionLC, 751
called by abbreviationsSpad2Cmd, 755
called by clearCmdParts, 773
called by clearSpad2Cmd, 767
called by close, 777
called by describeSpad2Cmd, 790
called by displaySpad2Cmd, 796
called by frameSpad2Cmd, 10
called by getTraceOption, 55
called by historySpad2Cmd, 822
called by newHelpSpad2Cmd, 813
called by readSpad2Cmd, 875
called by reportOpsFromLispLib, 1025
called by reportOpsFromUnitDirectly, 1028
called by setExposeAdd, 91
called by setExposeDrop, 94
called by setExpose, 91
called by setInputLibrary, 906
called by showHistory, 824
called by synonymsForUserLevel, 1038
called by systemCommand, 720
called by trace1, 48
called by unAbbreviateKeyword, 741
called by whatSpad2Cmd, 1091
called by workfilesSpad2Cmd, 1099
calls downcase, 751
calls object2Identifier, 751
calls selectOption, 751
defun, 751

sendHTErrorSignal
called by protectedEVAL, 271

separatePiles, 563
called by pileCforest, 562
called by separatePiles, 563
calls dqConcat, 563
calls dqUnit, 563
calls lastTokPosn, 563
calls separatePiles, 563
calls tokConstruct, 563
defun, 563

SEQ
called by funfind,LAM, 1060

seq
INDEX

called by /tracereply, 77
called by abbreviationsSpad2Cmd, 755
called by apropos, 1097
called by clearCmdParts, 773
called by coerceSpadArgs2E, 63
called by dewritify,dewritifyInner, 854
called by diffAlist, 1085
called by displayMacros, 799
called by displayProperties,sayFunctionDeps, 730
called by displayProperties, 733
called by flattenOperationAlist, 1061
called by getAliasIfTracedMapParameter, 75
called by getBpiNameIfTracedMap, 75
called by getPreviousMapSubNames, 1056
called by getTraceOption,ln, 54
called by getTraceOptions, 52
called by getTraceOption, 55
called by historySpad2Cmd, 822
called by importFromFrame, 19
called by isListOfIdentifiersOrStrings, 1056
called by isListOfIdentifiers, 1055
called by orderBySlotNumber, 1068
called by prTraceNames,fn, 79
called by prTraceNames, 80
called by recordFrame, 1083
called by removeUndoLines, 41
called by reportUndo, 43
called by runspad, 243
called by set1, 1014
called by spadReply,printName, 1069
called by spadReply, 1069
called by spadTrace,isTraceable, 66
called by spadTrace, 67
called by spadUntrace, 78
called by subTypes, 1055
called by trace1, 48
called by traceDomainConstructor, 71
called by traceReply, 81
called by undoFromFile, 836
called by undoLocalModemapHack, 40
called by undoSingleStep, 39
called by untraceDomainConstructor,keepTraceByFun, 73
called by untraceDomainConstructor, 73
called by whatConstructors, 1097
called by whatSpad2Cmd, 1091
called by writify,writifyInner, 849

called by serverReadLine, 268
called by intloopReadConsole, 253
calls addNewInterpreterFrame, 269
calls changeToNamedInterpreterFrame, 269
calls executeQuietCommand, 269
calls is-console[9], 269, 1159
calls lassoc, 269
calls mkprompt, 269
calls parseAndInterpret, 269
calls protectedEVAL, 269
calls read-line, 269
calls serverSwitch, 269
calls sockGetInt, 269
calls sockGetString, 269
calls sockSendInt, 269
calls sockSendString, 269
calls unescapeStringsInForm, 269
uses *eof*, 269
uses $CallInterp, 269
uses $CreateFrameAnswer, 269
uses $CreateFrame, 269
uses $EndOfOutput, 269
uses $EndServerSession, 269
uses $EndSession, 269
uses $KillLispSystem, 269
uses $LispCommand, 269
uses $MenuServer, 269
uses $NeedToSignalSessionManager, 269
uses $NonSmanSession, 269
uses $QuietSpadCommand, 269
uses $SessionManager, 269
uses $SpadCommand, 269
uses $SpadServer, 269
uses $SwitchFrames, 269
uses $currentFrameNum, 269
uses $frameAlist, 269
uses $frameNumber, 269
uses $sockBufferLength, 269
uses in-stream, 269
uses in-stream, 269
catches, 268, 269

called by serverSwitch
set, 892, 1013
  called by browse, 762
calls set1, 1013
uses $setOptions, 1014
defun, 1013
manpage, 892
set-hole-size
called by init-memory-config, 258
set-restart-hook, 237
defun, 237
set1, 1014
called by set1, 1015
called by set, 1013
calls bright, 1015
calls displaySetOptionInformation, 1014
calls displaySetVariableSettings, 1014
calls downcase, 1014
calls exit, 1014
calls lassoc, 1014
calls literals, 1015
calls object2String, 1015
calls poundsign, 1014
calls satisfiesUserLevel, 1014
calls sayKeyedMsg, 1014
calls sayMSG, 1014
calls selectOption, 1014
calls seq, 1014
calls setCurrentLine, 266
calls translateYesNo2TrueFalse, 1015
calls tree, 1015
calls use-fast-links, 1015
uses $UserLevel, 1015
uses $displaySetValue, 1015
uses $setOptionNames, 1015
defun, 1014
setAref2U16, 1252
defmacro, 1252
setAref2U32, 1253
defmacro, 1253
setAref2U8, 1250
defmacro, 1250
setCurrentLine, 266
called by intloopEchoParse, 294
called by intloopProcessString, 261
called by intloopProcess, 288
called by intloopReadConsole, 253
uses $currentLine, 266
defun, 266
setdatabase, 1155
called by abbreviationsSpad2Cmd, 755
calls make-database, 1155
defun, 1155
setdifference
called by displayWorkspaceNames, 727
called by untraceMapSubNames, 1059
setelt
called by computeTTTranspositions, 686
called by permuteToOrder, 687
called by setExposeAddConstr, 93
called by setExposeAddGroup, 89
called by setExposeDropConstr, 96
called by setExposeDropGroup, 94
seteltU16, 1248
defmacro, 1248
seteltU32, 1249
defmacro, 1249
seteltU8, 1247
defmacro, 1247
setExpose, 90
called by setExpose, 91
calls displayExposedConstructors, 90
calls displayExposedGroups, 90
calls displayHiddenConstructors, 90
calls namestring, 91
calls pathname, 91
calls qcar, 91
calls qcdr, 91
calls sayKeyedMsg, 91
calls sayMSG, 90
calls selectOptionLC, 91
calls setExposeAdd, 91
calls setExposeDrop, 91
calls setExpose, 91
defun, 90
setExposeAdd, 91
called by setExposeAdd, 92
called by setExpose, 91
calls displayExposedConstructors, 91
calls displayExposedGroups, 91
calls qcar, 91
calls qcdr, 91
INDEX

1825

calls sayKeyedMsg, 91
calls sayMSG, 91
calls selectOptionLC, 91
calls setExposeAddConstr, 92
calls setExposeAddGroup, 92
calls setExposeAdd, 92
calls specialChar, 91
uses $linelength, 92
defun, 91

setExposeAddConstr, 92
calledby localnlib, 1161
calledby setExposeAdd, 92
calls clearClams, 93
calls delete, 93
calls displayExposedConstructors, 93
calls getdatabase, 92
calls member, 92
calls msort, 93
calls qcar, 92
calls sayKeyedMsg, 92
calls setelt, 93
calls specialChar, 93
calls unabbrev, 92
uses $interpreterFrameName, 93
uses $linelength, 93
uses $localExposureData, 93
defun, 92

setExposeAddGroup, 89
calledby setExposeAdd, 92
calls clearClams, 89
calls displayExposedConstructors, 89
calls displayExposedGroups, 89
calls displayHiddenConstructors, 89
calls getalist, 89
calls member, 89
calls msort, 89
calls namestring, 89
calls object2String, 89
calls pathname, 89
calls qcar, 89
calls sayAsManyPerLineAsPossible, 89
calls sayKeyedMsg, 89
calls sayMSG, 89
calls setelt, 89
calls specialChar, 89
uses $globalExposureGroupAlist, 89
uses $interpreterFrameName, 89
uses $linelength, 89
uses $localExposureData, 89
defun, 89

calls setExposeDrop, 93
calledby setExposeDrop, 94
calledby setExpose, 91
calls displayHiddenConstructors, 93
calls qcar, 94
calls qcdr, 94
calls sayKeyedMsg, 94
calls sayMSG, 94
calls selectOptionLC, 94
calls setExposeDropConstr, 94
calls setExposeDropGroup, 94
calls setExposeDrop, 94
calls specialChar, 93
uses $linelength, 94

defun, 93

setExposeDropConstr, 96
calledby setExposeDrop, 94
calls clearClams, 96
calls delete, 96
calls displayExposedConstructors, 96
calls displayHiddenConstructors, 96
calls getdatabase, 96
calls member, 96
calls msort, 96
calls qcar, 96
calls sayKeyedMsg, 96
calls sayMSG, 96
calls setelt, 96
calls specialChar, 96
calls unabbrev, 96
uses $interpreterFrameName, 96
uses $linelength, 96
uses $localExposureData, 96
defun, 96

setExposeDropGroup, 94
calledby setExposeDrop, 94
calls clearClams, 95
calls delete, 95
calls displayExposedConstructors, 94
calls displayExposedGroups, 94
calls displayHiddenConstructors, 95
calls getalist, 95
calls member, 95
calls qcar, 94
calls sayKeyedMsg, 95
calls sayMSG, 94
calls setelt, 94
calls specialChar, 95
uses $globalExposureGroupAlist, 95
uses $interpreterFrameName, 95
uses $linelength, 95
uses $localExposureData, 95
defun, 94

setFortDir, 930
calls bright, 931
calls describeSetFortDir, 930
calls pname, 930
calls sayBrightly, 930
calls validateOutputDirectory, 930
uses $fortranDirectory, 931
defun, 930

setFortPers, 961
calls bright, 961
calls describeFortPersistence, 961
calls sayMessage, 961
calls terminateSystemCommand, 961
uses $fortPersistence, 961
defun, 961

setFortTmpDir, 928
calls bright, 928
calls describeSetFortTmpDir, 928
calls pname, 928
calls sayBrightly, 928
calls validateOutputDirectory, 928
uses $fortranTmpDir, 928
defun, 928

setFunctionsCache, 911
calls bright, 911
calls countCache, 911
calls describeSetFunctionsCache, 911
calls object2String, 911
calls sayAllCacheCounts, 911
calls sayMessage, 911
calls terminateSystemCommand, 911
uses $cacheAlist, 912
uses $cacheCount, 912
uses $options, 911
defun, 911

setHistoryCore, 826
calledby historySpad2Cmd, 822
calls boot-equal, 826
calls disableHist, 826
calls histFileErase, 826
calls histFileName, 826
calls object2Identifier, 826
calls rdefiostream, 826
calls readHiFi, 826
calls rkeyids, 826
calls rshut, 826
calls sayKeyedMsg, 826
calls spadrwrite, 826
uses $HiFiAccess, 826
uses $IOindex, 826
uses $internalHistoryTable, 826
uses $useInternalHistoryTable, 826
defun, 826

setInputLibrary, 906
calledby setInputLibrary, 906
calls addInputLibrary, 906
calls describeInputLibraryArgs, 906
calls dropInputLibrary, 906
calls qcar, 906
calls qcdr, 906
calls selectOptionLC, 906
calls setInputLibrary, 906
uses input-libraries, 906
defun, 906

setIOindex, 841
calledby showHistory, 824
uses $IOindex, 841
defun, 841

setletprintflag
calledby breaklet, 86
calledby spadTrace, 67
calledby spadUntrace, 78
calledby tracelet, 85

setLinkerArgs, 933
calls describeSetLinkerArgs, 933
calls object2String, 933
uses $fortranLibraries, 933
defun, 933

setMsgCatlessAttr, 587
calledby setMsgForcedAttr, 586
calledby setMsgUnforcedAttr, 589
calls ifcdr, 587
  calls ncAlist, 587
calls ncPutQ, 587
calls qassq, 587
defun, 587
setMsgForcedAttr, 586
calledby setMsgForcedAttrList, 586
calls ncPutQ, 586
calls setMsgCatlessAttr, 586
defun, 586
setMsgForcedAttrList, 586
calledby msgCreate, 570
calls whichCat, 586
defun, 586
setMsgPrefix, 571
calledby processChPosesForOneLine, 598
defmacro, 571
setMsgText, 572
calledby putDatabaseStuff, 587
calledby putFTTText, 601
defmacro, 572
setMsgUnforcedAttr, 589
calledby initImPr, 589
calledby initToWhere, 590
calledby setMsgUnforcedAttrList, 588
calls ncAlist, 589
calls ncPutQ, 589
calls qassq, 589
calls setMsgCatlessAttr, 589
defun, 589
setMsgUnforcedAttrList, 588
calledby putDatabaseStuff, 587
calls setMsgUnforcedAttr, 588
calls whichCat, 588
defun, 588
setNagHost, 959
calls describeSetNagHost, 959
calls object2String, 959
uses $nagHost, 959
defun, 959
setOutputAlgebra, 967
calledby describeSetOutputAlgebra, 969
calledby spad, 242
calls $filep, 967
calls concat, 967
calls defostream, 967
calls describeSetOutputAlgebra, 967
calls make-outstream, 967
calls member, 967
calls object2String, 967
calls pathnameDirectory, 967
calls pathnameName, 967
calls pathnameType, 967
calls qcar, 967
calls qcdr, 967
calls sayKeyedMsg, 967
calls shut, 967
calls upcase, 967
uses $algebraFormat, 967
uses $algebraOutputFile, 967
uses $algebraOutputStream, 967
uses $filep, 967
defun, 967
setOutputCharacters, 971
calledby setOutputCharacters, 971
calls bright, 971
calls concat, 971
calls downcase, 971
calls pname, 971
calls qcar, 971
calls qcdr, 971
calls sayAsManyPerLineAsPossible, 971
calls sayBrightly, 971
calls sayMessage, 971
calls setOutputCharacters, 971
calledby setOutputCharacters, 971
calls specialChar, 971
uses $RTspecialCharacters, 971
uses $plainRTspecialCharacters, 971
uses $specialCharacterAlist, 971
uses $specialCharacters, 971
defun, 971
setOutputFormula, 995
calledby describeSetOutputFormula, 998
calls $filep, 995
calls concat, 995
calls defostream, 995
calls describeSetOutputFormula, 995
calls make-outstream, 995
calls member, 995
calls object2String, 995
calls pathnameDirectory, 995
calls pathnameName, 995
calls pathnameType, 995
calls qcar, 995
calls qcdr, 995
calls sayKeyedMsg, 995
calls shut, 995
calls upcase, 995
uses $lep, 996
uses $formulaFormat, 996
uses $formulaOutputStream, 996
defun, 995
setOutputFortran, 974
calledby describeSetOutputFortran, 977
calls $filep, 974
calls concat, 974
calls defostream, 974
calls describeSetOutputFortran, 974
calls makeStream, 974
calls member, 974
calls object2String, 974
calls pathnameDirectory, 974
calls pathnameName, 974
calls pathnameType, 974
calls qcar, 974
calls qcdr, 974
calls sayKeyedMsg, 974
calls shut, 974
calls upcase, 974
uses $filep, 974
uses $fortranFormat, 974
uses $fortranOutputStream, 974
defun, 974
setOutputHtml, 980
calledby describeSetOutputHtml, 982
calls $filep, 980
calls concat, 980
calls defostream, 980
calls describeSetOutputHtml, 980
calls make-outstream, 980
calls member, 980
calls object2String, 980
calls pathnameDirectory, 980
calls pathnameName, 980
calls pathnameType, 980
calls qcar, 980
calls qcdr, 980
calls sayKeyedMsg, 980
calls shut, 980
calls upcase, 980
uses $filep, 980
uses $htmlFormat, 980
uses $htmlOutputFile, 980
uses $htmlOutputStream, 980
defun, 980
setOutputLibrary, 904
calledby describeOutputLibraryArgs, 904
calls $filep, 904
calls openOutputLibrary, 904
calls poundsign, 904
uses $outputLibraryName, 904
defun, 904
setOutputMathml, 986
calledby describeSetOutputMathml, 988
calls $filep, 986
calls concat, 986
calls defostream, 986
calls describeSetOutputMathml, 986
calls make-outstream, 986
calls member, 986
calls object2String, 986
calls pathnameDirectory, 986
calls pathnameName, 986
calls pathnameType, 986
calls qcar, 986
calls qcdr, 986
calls sayKeyedMsg, 986
calls shut, 986
calls upcase, 986
uses $filep, 986
uses $mathmlFormat, 986
uses $mathmlOutputFile, 986
uses $mathmlOutputStream, 986
defun, 986
setOutputOpenMath, 990
calledby describeSetOutputOpenMath, 993
calls $filep, 991
calls concat, 990
calls defostream, 990
calls describeSetOutputOpenMath, 990
calls make-outstream, 991
calls member, 990
calls object2String, 991
calls pathnameDirectory, 991
calls pathnameName, 991
calls pathnameType, 991
calls qcar, 990
calls qcdr, 990
calls sayKeyedMsg, 991
calls shut, 991
uses $filep, 991
uses $openMathFormat, 991
uses $openMathOutputStream, 991
defun, 990
setOutputTex, 1001
called by describeSetOutputTex, 1004
calls $filep, 1002
calls concat, 1002
calls defostream, 1001
calls describeSetOutputTex, 1002
calls make-ostream, 1002
calls member, 1002
calls object2string, 1002
calls pathnameDirectory, 1002
calls pathnameName, 1002
calls pathnameType, 1002
calls qcar, 1002
uses $filep, 1002
uses $openMathFormat, 1002
uses $openMathOutputStream, 1002
defun, 1001
setStreamsCalculate, 1007
calls bright, 1007
calls describeSetStreamsCalculate, 1007
calls object2string, 1007
calls sayMessage, 1007
calls terminateSystemCommand, 1007
uses $streamCount, 1007
defun, 1007
setUpDefault, 1428
defun, 1428
shortenForPrinting, 1067
called by letPrint, 1062
calls devaluate, 1067
calls isDomainOrPackage, 1067
defun, 1067
show, 1018, 1019
calls showSpad2Cmd, 1019
defun, 1019
manpage, 1018
showdatabase, 1154
calls getdatabase, 1154
defun, 1154
showHistory, 824
called by historySpad2Cmd, 822
calls bright, 824
calls concat, 824
calls sayKeyedMsg, 824
calls sayMSG, 824
calls selectOptionLC, 824
calls setIOindex, 824
calls showInOut, 824
calls showInput, 824
calls showSpad2Cmd, 824
calls $evalTimePrint, 825
uses $printTimeSum, 825
defun, 824
showInOut, 842
called by showHistory, 824
calls asq, 842
calls disableHist, 842
calls objMode, 842
calls objValUnwrap, 842
calls readHiFi, 842
calls sayMSG, 842
calls spadPrint, 842
defun, 842
showInput, 841
called by showHistory, 824
calls disableHist, 841
calls readHiFi, 841
calls sayMSG, 841
calls tab, 841
defun, 841
showMsgPos?, 580
called by getPosStL, 578
calls leader?, 580
calls msgImPr?, 580
uses $erMsgToss, 580
defun, 580
showSpad2Cmd, 1019
calledby show, 1019
calls helpSpad2Cmd, 1019
calls member, 1019
calls qcar, 1019
calls reportOperations, 1019
calls sayKeyedMsg, 1019
calledby setOutputFormula, 1019
calledby setOutputFortran, 1019
calledby setOutputOpenMath, 1019
calledby setOutputTex, 1019
calledby writeInputLines, 1019
calledby is-console[9], 1128
defun, 1128
shut, 1128
  calledby reportOpsFromLisplib1, 1024
calledby reportOpsFromUnitDirectly1, 1032
calledby setOutputAlgebra, 967
calledby setOutputFormula, 995
calledby setOutputFortran, 974
calledby setOutputHtml, 980
calledby setOutputMathml, 986
calledby setOutputOpenMath, 991
calledby setOutputTex, 1002
calledby writeInputLines, 1019
calledby next1, 263
calledby serverReadLine, 269
calledby set-restart-hook, 237
calledby setCurrentLine, 266
StreamNull, 555
simpCatPredicate
  calledby dbSearchOrder, 1514
simpHasPred
  calledby domainDescendantsOf, 1507
calledby kTestPred, 1543
calledby kePage, 1510
calledby reduceAlistForDomain, 1520
size, 1196
  calledby abbreviationsSpad2Cmd, 755
calledby getPreStL, 577
calledby isgenvar, 1064
calledby makeMsgFromLine, 593
calledby processChPosesForOneLine, 598
calledby processSynonyms, 257
calledby substringMatch, 342
calledby writeInputLines, 828
defun, 1196
SkipEnd?, 305
  calledby incLude1, 307
calledby remainder, 305
defun, 305
SkipPart?, 305
  calledby incLude1, 307
calledby remainder, 305
defun, 305
Skipping?, 306
  calledby incLude1, 306
calledby incLude1, 306
calledby KeepPart?, 306
defun, 306
sockGetInt
  calledby queryClients, 777
calledby serverReadLine, 269
sockGetString
  calledby executeQuietCommand, 272
calledby serverReadLine, 269
sockSendInt
  calledby close, 777
calledby queryClients, 777
calledby serverReadLine, 269
sockSendString
  calledby serverReadLine, 269
INDEX

sortby
called by workfilesSpad2Cmd, 1099

spad, 242
called by restart, 239
calls runspad, 242
calls setOutputAlgebra, 242
uses $PrintCompilerMessageIfTrue, 242
defun, 242

spad-error-loc, 1116
called by spad-long-error, 1115
defun, 1116

spad-long-error, 1115
called by spad-syntax-error, 1115
calls iostat, 1115
calls spad-error-loc, 1115
uses out-stream, 1115
uses spaderrorstream, 1115
defun, 1115

spad-save, 1135
called by savesystem, 891
calls save-system, 1136
uses $SpadServer, 1136
uses $openServerIfTrue, 1136
defun, 1135

spad-short-error, 1116
called by spad-syntax-error, 1115
calls line-past-end-p, 1116
calls line-print, 1116
uses $current-line, 1116
defun, 1116

spad-syntax-error, 1115
calls bumperrorcount, 1115
calls consoleinputp, 1115
calls ioclear, 1115
calls spad-error-loc, 1115
calls spad-long-error, 1115
uses debugmode, 1115
defun, 1115

spad2BootCoerce, 1205
defun, 1205

spadcall
called by algEqual, 696
called by basicLookupCheckDefaults, 1188
called by basicLookup, 1185
called by clearCmdSortedCaches, 769
called by getConstantFromDomain, 699
called by setPrint3, 1066
called by lookupInDomainVector, 1187
called by retractByFunction, 710

spadClosure?, 853
called by writify, writifyInner, 849
calls bpiname, 853
calls qcar, 853
calls qcdr, 853
calls vecp, 853
defun, 853

spadConstant, 1305
defmacro, 1305

spaderrorstream
used by init-boot/spad-reader, 1114
used by spad-long-error, 1115

SpadInterpretStream, 250
called by intloop, 250
calls intloopInclude, 252
calls intloopReadConsole, 252
calls mkprompt, 252
uses $erMsgToss, 252
uses $fn, 252
uses $inclAssertions, 252
uses $lastPos, 252
uses $libQuiet, 252
uses $ncMsgList, 252
uses $newcompErrorCount, 252
uses $npos, 252
uses $okToExecuteMachineCode, 252
uses $systemCommandFunction, 252
defun, 250

spadPrint
called by showInOut, 842

spadReply, 1069
called by spadTrace, 67
called by spadUntrace, 78
calls exit, 1069
calls seq, 1069
calls spadReply,printName, 1069
uses /tracenames, 1069
defun, 1069

spadReply,printName, 1069
called by spadReply, 1069
calls devaluate, 1069
calls exit, 1069
calls isDomainOrPackage, 1069
calls qcar, 1069
calls seq, 1069
defun, 1069
SPADRREAD
calls dewritify, 847
calls rread, 847
spadrread, 847
called by readHiFi, 843
defun, 847
spadrwrite, 847
called by setHistoryCore, 826
called by writeHiFi, 844
calls spadrwrite0, 847
calls throwKeyedMsg, 847
defun, 847
spadrwrite0, 846
called by saveHistory, 837
called by spadrwrite, 847
calls rwrite, 846
calls safeWritify, 846
defun, 846
spadStartUpMsgs, 241
called by restart, 239
calls fillerSpaces, 241
calls sayKeyedMsg, 241
calls sayMSG, 241
calls specialChar, 241
uses *build-version*, 241
uses *yearweek*, 241
uses $linelength, 241
uses $msgAlist, 241
uses $opSysName, 241
defun, 241
spadSysname

called by coerceTraceArgs2E, 61
called by coerceTraceFunValue2E, 64
spadThrow

called by pf2Sex1, 531
called by tersyscommand, 724
called by throwEvalTypeMsg, 1082
spadTrace, 67
called by traceDomainConstructor, 71
calls aldorTrace, 67
calls as-insert, 67
calls assoc, 67
calls bpiname, 67
calls bptrace, 67
calls constructSubst, 67
calls exit, 67
calls flattenOperationAlist, 67
calls getOperationAlistFromLisplib, 67
calls option, 67
calls isDomainOrPackage, 67
calls opOf, 67
calls printDashedLine, 67
calls removeOption, 67
calls reportSpadTrace, 67
calls rplace, 67
calls seq, 67
calls setletprintflag, 67
calls spadReply, 67
calls spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
spadTraceAlias, 76
called by spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
called by spadTrace, 67
calls bpiname, 78
calls bpinuntrace, 78
calls bright, 78
calls delasc, 78
calls devaluate, 78
calls exit, 78
calls getOption, 78
calls isDomainOrPackage, 77
calls prefix2String, 78
calls remover, 78
calls rplac, 78
calls sayMSG, 78
calls seq, 78
calls setletprintag, 78
calls spadReply, 78
calls userError, 78
uses /tracenames, 78
uses $letAssoc, 78
defun, 77
spleI, 345
calledby displayOperationsFromLisplib, 1027
calledby displaySetOptionInformation, 896
calledby displaySetVariableSettings, 897
calledby filterAndFormatConstructors, 1096
calledby printSynonyms, 745
calledby reportOpsFromLisplib, 1025
calledby reportOpsFromUnitDirectly, 1028
calledby setExposeAddConstr, 93
calledby setExposeAddGroup, 89
calledby setExposedAdd, 91
calledby setExposeDropConstr, 96
calledby setExposedDropGroup, 95
calledby setExposedDrop, 93
calledby setOutputCharacters, 971
calledby spadStartUpMsgs, 241
calledby whatCommands, 1093
calledby workfilesSpad2Cmd, 1099
calls assq, 1126
calls ifcdr, 1126
uses $specialCharacterAlist, 1126
uses $specialCharacters, 1126
defun, 1126
spleI, 346
calledby scanPossFloat, 344
calledby spleI, 345
defun, 345
split, 885
calledby testpassed, 885
calls endedp, 886
calls ignorep, 886
calls resultp, 886
calledby startp, 886
defun, 885
spool, 1033
manpage, 1033
stackTraceOptionError, 1054
calledby getTraceOption.hn, 54
called by getTraceOption, 55
called by traceOptionError, 59
called by transOnlyOption, 1053
uses $traceErrorStack, 1054
defun, 1054
startp, 887
called by split, 886
called by testnumberp, 884
calls lastcount, 887
uses *ok*, 887
defun, 887
startsComment?, 339
called by scanEsc, 347
called by scanToken, 337
uses $ln, 339
uses $n, 339
uses $sz, 339
defun, 339

startsId?, 1194
called by scanToken, 337
defmacro, 1194

startsNegComment?, 340
called by scanEsc, 347
called by scanToken, 337
uses $ln, 340
uses $n, 340
uses $sz, 340
defun, 340

startTimingProcess
called by coerceInteractive, 672
called by evalDomain, 1075
called by loadLib, 1181
called by localnrlib, 1161
called by processInteractive1, 277
called by recordNewValue, 833
called by recordOldValue, 833
called by updateHist, 831

statisticsInitialization, 1181
called by restart, 239
calls gbc-time, 1181
defun, 1181

statisticsSummary
called by printStatisticsSummary, 283

stopTimingProcess
called by coerceInteractive, 672
called by evalDomain, 1075
called by interpretTopLevel, 278
called by loadLibNoUpdate, 1183
called by loadLib, 1182
called by processInteractive1, 277
called by recordNewValue, 833
called by recordOldValue, 833
called by updateHist, 831

strconc
called by bcComplexLimitGen, 1379
called by bcDefiniteIntegrateGen, 1345
called by bcDifferentiateGen, 1350
called by bcDraw2DSolveGen, 1357
called by bcDraw2DfunGen, 1353
called by bcDraw2DparGen, 1355
called by bcDraw3DfunGen, 1359
called by bcDraw3Dpar1Gen, 1363
called by bcDraw3DparGen, 1361
called by bcDrawIt, 1401
called by bcGenEquations, 1396
called by bcGen, 1398, 1399
called by bcIndefiniteIntegrateGen, 1344
called by bcInputEquations, 1385
called by bcInputExplicitMatrix, 1339
called by bcInputMatrixByFormulaGen, 1339
called by bcLinearMatrixGen, 1394
called by bcLinearSolveMatrixInhomo, 1392
called by bcMakeEquations, 1388
called by bcMakeLinearEquations, 1388
called by bcMatrixGen, 1341
called by bcProductGen, 1348
called by bcRealLimitGen1, 1377
called by bcSeriesExpansionGen, 1366
called by bcSeriesGen, 1373
called by bcSolveEquations, 1390
called by bcSumGen, 1347
called by bcwords2liststring, 1400
called by coerceIntCommute, 689
called by consSpecialString?, 1502
called by dbWordFrom, 1496
called by displayMacro, 726
called by displayValue, 731
called by editFile, 805
called by fixObjectForPrinting, 728
called by htMkName, 1402
called by htStringPad, 1402

called by kPage, 1497
called by makeMsgFromLine, 593
called by processChiPosesForOneLine, 598
called by processSynonyms, 257
called by reportOpsFromLisplib, 1024
called by reportOpsFromUnitDirectly, 1028
called by whatCommands, 1093

streamChop, 297
called by ncloopDQlines, 296
called by streamChop, 297
calls StreamNull, 297
calls ncloopPrefix?, 297
calls streamChop, 297
defun, 297
StreamNil, 329
used by incRgen1, 330
defvar, 329
StreamNull, 555
called by incAppend1, 312
called by incLude1, 306
called by incZip1, 299
called by intloopProcess, 288
called by ncloopDQlines, 296
called by next1, 263
called by npNull, 555
called by parseFromString, 273
called by parseNoMacroFromString, 1531
called by streamChop, 297
calls eqcar, 555
defun, 555
string-concatenate
called by concat, 1197
string2Constructor, 1494
called by conSpecialString?, 1502
calls downcase, 1494
calls hget, 1494
calls ifcar, 1494
uses $lowerCaseConTb, 1494
defun, 1494
string2id-n
called by close, 777
called by historySpad2Cmd, 822
called by importFromFrame, 19
called by listConstructorAbbreviations, 756
strpos, 1196
called by processSynonyms, 257
called by quitSpad2Cmd, 872
called by synonymsForUserLevel, 1038
called by yesanswer, 798
string2Integer

called by dbGetDocTable, 1542
string2pint-n
called by getAliasIfTracedMapParameter, 74
string2Words

called by conSpecialString?, 1502
called by segmentKeyedMsg, 28
stringize, 1415
defun, 1415
stringList2String, 1402
defun, 1402
stringMatches?, 1210
calls basicMatch?, 1210
defun, 1210
stringPrefix?, 1334
called by clearCmdExcept, 772
called by hasOption, 723
called by removeUndoLines, 41
called by selectOption, 751
called by undo, 36
defun, 1334
stringSpaces
called by getFirstWord, 741
StringToDir, 1228
called by fnameMake, 1228
calls lastc, 1228
defun, 1228
stripLisp, 743
called by handleNoParseCommands, 742
defun, 743
stripSpaces, 743
called by dumbTokenize, 739
called by handleNoParseCommands, 742
called by parseSystemCmd, 740
called by splitIntoOptionBlocks, 719
defun, 743
stripUnionTags, 708
called by coerceBranch2Union, 698
called by coerceUnion2Branch, 707
defun, 708
strposl, 1196
called by getSystemCommandLine, 1038
called by processSynonyms, 257
called by rdigit?, 356
called by satisfiesRegularExpressions, 1095
called by scanS, 354
called by stupidIsSpadFunction, 1070
defun, 1196
stupidIsSpadFunction, 1070
called by breaklet, 86
called by tracelet, 85
calls pname, 1070
calls strpos, 1070
defun, 1070
called by domArg2, 652
called by hasAtt, 654
called by hasCate, 663
called by hasCaty, 650
called by hasSigAnd, 655
called by hasSig, 653
called by replaceSharps, 1104
called by unifyStructVar, 659
sublisFormal
   called by dbShowConsDoc1, 1550
called by kArgPage, 1506
called by kePage, 1509
sublisIs
   called by dbAddDocTable, 1541
called by dbConstructorDoc, hn, 1538
called by dbGetDocTable, hn, 1541
called by dbSearchOrder, 1514
called by dbShowConsDoc1, 1550
called by kcnPage, 1526
called by kcpPage, 1519
called by libConstructorSig, 1563
called by localnrlib, 1161
called by mkDomTypeForm, 1507
called by reduceAlistForDomain, 1519
subMatch, 342
   called by scanPunct, 341
calls substringMatch, 342
defun, 342
subseq
   called by getFirstWord, 741
called by kdPageInfo, 1505
substFromAlist, 1433
defun, 1433
substituteSegmentedMsg
   called by getMsgInfoFromKey, 588
called by msgText, 286
called by sayKeyedMsgLocal, 28
substring, 256
   called by ExecuteInterpSystemCommand, 256
called by alqlGetKindString, 1236
called by alqlGetOrigin, 1235
called by alqlGetParams, 1236
called by doSystemCommand, 718
called by getAliasIfTracedMapParameter, 75
called by getSystemCommandLine, 1038
called by incDrop, 327
called by lineoftoks, 335
called by mkprompt, 267
called by printLabelledList, 746
called by processSynonyms, 257
called by removeUndoLines, 41
called by scanComment, 339
called by scanNegComment, 340
called by scanS, 354
called by scanW, 351
called by spleI1, 346
called by writeInputLines, 828
defun, 256
substringMatch, 342
called by subMatch, 342
calls size, 342
defun, 342
subTypes, 1055
called by spadTrace, 67
called by subTypes, 1055
calls exit, 1055
calls lassoc, 1055
calls seq, 1055
calls subTypes, 1055
defun, 1055
suchthat, 628
syntax, 628
summary, 1034, 1035
   calls concat, 1035
called by getenviron, 1035
called by obey, 1035
defun, 1035
manpage, 1034
superMatch?
called by dbShowCons, 1545
called by koaPageFilterByName, 1538
syGeneralErrorHere, 416
called by npListAndRecover, 414
called by npTrapForm, 437
calls sySpecificErrorHere, 416
defun, 416
syIgnoredFromTo, 416
called by npRecoverTrap, 415
INDEX

1837

calls FromTo, 416
calls From, 416
calls To, 416
calls ncSoftError, 416
calls pfGlobalLinePosn, 416
defun, 416

synonym, 1036, 1037

calls synonymSpad2Cmd, 1037
defun, 1037

manpage, 1036

synonymsForUserLevel, 1038

called by printSynonyms, 745
calls commandsForUserLevel, 1038
calls selectOptionLC, 1038
calls string2id-n, 1038
uses $UserLevel, 1038
uses $systemCommands, 1038
defun, 1038

synonymSpad2Cmd, 1037

called by synonym, 1037
calls getSystemCommandLine, 1037
calls printSynonyms, 1037
calls processSynonymLine, 1037
calls putalist, 1037
calls terminateSystemCommand, 1037
uses $CommandSynonymAlist, 1037
defun, 1037

syntax

assignment, 605
blocks, 608
clef, 610
collection, 611
for, 613
if, 617
iterate, 619
leave, 620
parallel, 621
repeat, 624
suchthat, 628
while, 629

sySpecificErrorAtToken, 417

called by sySpecificErrorHere, 416
calls ncSoftError, 417
calls tokPosn, 417
defun, 417

sySpecificErrorHere, 416
called by syGeneralErrorHere, 416
calls sySpecificErrorAtToken, 416
uses $stok, 417

defun, 416

system, 1041

manpage, 1041

systemCommand, 720

called by handleParsedSystemCommands, 740
called by handleTokenizeSystemCommands, 719
calls commandsForUserLevel, 720
calls helpSpad2Cmd, 720
calls selectOptionLC, 720
calls selectOption, 720
uses $CategoryFrame, 720
uses $e, 720
uses $options, 720
uses $syscommands, 720
uses $systemCommands, 720
defun, 720

systemError

called by bcInputEquationsEnd, 1389
called by bcMatrixGen, 1341
called by dbShowConsDoc, 1549
called by domainDescendantsOf, 1507
called by explainLinear, 1395
called by koPageAux, 1537
called by mkConform, 1532
called by pfDefinition2Sex, 545
called by pfLambdaTran, 546

systemErrorHere

called by interpret2, 280
called by reportOpsFromUnitDirectly, 1028

tab

called by showInput, 841

tabbing, 584
called by getStFromMsg, 577
calls getMsgPrefix?, 584
uses $preLength, 584
defun, 584

take

called by 8?, 84
called by libConstructorSig, 1563

tangle, 1042
manpage, 1042
templateParts, 1437
defun, 1437
terminateSystemCommand, 724
calledby commandAmbiguityError, 724
calledby commandErrorMessage, 722
calledby displayProperties, 733
calledby npProcessSynonym, 745
calledby setFortPers, 961
calledby setFunctionsCache, 911
calledby synonymSpad2Cmd, 1037
calledby setUserLevelErrorMessage, 723
calls tersyscommand, 724
defun, 724
tersyscommand, 724
calledby library, 1159
calledby quitSpad2Cmd, 872
calledby terminateSystemCommand, 724
calls spadThrow, 724
defun, 724
testBitVector
calledby kTestPred, 1543
testnumberp, 884
calledby findnexttest, 884
calls startp, 884
defun, 884
testpassed, 885
calledby regress, 882
calls split, 885
defun, 885
The restart function, 238
thefname, 316
calledby inclmsgCannotRead, 317
calledby inclmsgNoSuchFile, 316
called by pfname, 316
defun, 316
theid, 315
called by inclmsgConActive, 318
called by inclmsgConStill, 319
called by inclmsgFileCycle, 317
called by inclmsgIfSyntax, 322
called by inclmsgSay, 315
defun, 315
theorigin, 313
calledby inclmsgIfSyntax, 322
calledby inclmsgPrematureEOF, 313
calledby inclmsgPrematureFin, 321
defun, 313
thisPosIsEqual, 596
calls poGlobalLinePosn, 596
defun, 596
thisPosIsLess, 596
calls poGlobalLinePosn, 596
defun, 596
throwEvalTypeMsg, 1082
calledby evaluateType1, 1080
calledby evaluateType, 1079
calls spadThrow, 1082
calls throwKeyedMsg, 1082
local ref $noEvalTypeMsg, 1082
defun, 1082
throwKeyedMsg
calledby addNewInterpreterFrame, 18
calledby closeInterpreterFrame, 22
calledby close, 777
calledby coerceInteractive, 672
calledby fetchOutput, 842
calledby frameSpad2Cmd, 10
calledby getConstantFromDomain, 699
calledby getTraceOptions, 52
calledby getTraceOption, 55
calledby importFromFrame, 19
calledby readSpad2Cmd, 875
calledby restoreHistory, 839
calledby saveHistory, 837
calledby spadrwrite, 847
calledby throwEvalTypeMsg, 1082
calledby trace1, 48
calledby transTraceItem, 60
calledby workfilesSpad2Cmd, 1099
calledby writeInputLines, 828
throwKeyedMsgCannotCoerceWithValue
calledby catchCoerceFailure, 692
calledby evaluateType1, 1080
calledby interpret2, 280
throwListOfKeyedMsgs
calledby getTraceOptions, 52
throws
coercionFailure, 695
fin, 808
intloopReadConsole, 253
monitor-file, 1319
monitor-readinterp, 1326
monitor-spadfile, 1328
ncError, 293
npMissing, 375
npTrap, 437
npTrapForm, 437
ScanOrPairVec, ScanOrInner, 857
spad-syntax-error, 1115
writify, writifyInner, 849
timedEVALFUN
called by getAndEvalConstructorArgument, 1104
timedEvaluate
called by coerceIntByMap, 693
tmp1
called by diffAlist, 1085
To, 602
called by syIgnoredFromTo, 416
defun, 602
toFile?, 585
called by msgOutputter, 576
calls getMsgToWhere, 585
uses $fn, 585
defun, 585
tokConstruct, 633
called by enPile, 562
called by npDDInfKey, 432
called by npDollar, 408
called by npFirstTok, 367
called by npId, 429
called by npInfixOperator, 384
called by npPrefixColon, 385
called by npPushId, 433
called by npSymbolVariable, 430
called by pfLeaf, 477
called by separatePiles, 563
calls icar, 633
calls ncPutQ, 633
calls pfNoPosition?, 633
defun, 633
token-stack-show, 1117
called by iostat, 1116
calls type-token, 1117
uses current-token, 1117
uses next-token, 1117
uses prior-token, 1117
uses valid-tokens, 1117
defun, 1117
tokPart, 635
called by ntooopProcess, 288
called by npDDInfKey, 432
called by npFirstTok, 367
called by npInfixOperator, 384
called by npSymbolVariable, 430
called by pf2Sex1, 532
called by pfCopyWithPos, 466
called by pfIdSymbol, 477
called by(pfLeafToken, 478
called by pfLiteralString, 479
called by pfSexpr,strip, 480
called by pfSymbolSymbol, 482
called by pfSymb, 481
called by pileCforest, 562
defun, 635
tokPosn, 635
called by firstTokPosn, 563
called by lastTokPosn, 563
called by ncloopDQlines, 296
called by npConstTok, 409
called by npDDInfKey, 432
called by npDollar, 408
called by npFirstTok, 367
called by npId, 429
called by npInfixOperator, 384
called by npMissingMate, 440
called by npMissing, 376
called by npParse, 365
called by npPrefixColon, 386
called by npPushId, 433
called by npRecoverTrap, 415
called by npSymbolVariable, 430
called by npTrapForm, 437
called by npTrapp, 437
called by pfBraceBar, 487
called by pfBrace, 487
called by pfBracketBar, 488
called by pfBracket, 487
called by pfLeafPosition, 478
called by pileColumn, 559
called by sysSpecificErrorAtToken, 417
calls ncAlist, 635
calls pfNoPosition, 635
calls qassq, 635
defun, 635
tokTran, 739
called by handleParsedSystemCommands, 740
called by handleTokensizeSystemCommands, 719
called by parseSystemCmd, 740
calls isIntegerString, 739
defun, 739
tokType, 635
called by npConstTok, 408
called by pilePlusComment, 558
calls ncTag, 635
defun, 635
Top, 302
  used by incStream, 298
  used by incString, 263
defvar, 302
Top?, 304
called by incLude1, 306
called by xlIfSyntax, 322
calls quotient, 304
defun, 304
toplevel
called by loadLibNoUpdate, 1183
topLevelInterpEval, 1530
calls processInteractive, 1530
  uses $ProcessInteractiveValue, 1530
  uses $noEvalTypeMsg, 1530
defun, 1530
toScreen?, 573
called by msgOutputter, 576
calls getMsgToWhere, 573
defmacro, 573
TPDHERE
  Beware that this function occurs with lowercase also, 389
  Beware that this function occurs with uppercase also, 389

Make this more international, not EBCDIC, 1126
Note that there is also an npADD function, 383
Note that there is also an npAdd function, 382
Remove all boot references from top level, 744
The pform function has a leading percent sign. fix this, 452, 453, 457
The pform function has a leading percent sign, 291, 449
The variable al is undefined, 587
This could probably be replaced by the default assoc using eql, 1200
This function should be replaced by fillerspaces, 593
This should probably be a macro or eliminated, 743
Well this makes no sense., 421
getMsgFTTag is nonsense, 600
note that the file interp.exposed no longer exists., 1326
rewrite this using (dolist (item seqList)...), 540
rewrite using dsetq, 541
trace, 47, 1045
called by ltrace, 868
calls traceSpad2Cmd, 47
defun, 47
manpage, 1045
trace1, 48
called by trace1, 48
called by traceSpad2Cmd, 47
calls /trace,0, 49
calls ?t, 48
calls addassoc, 49
calls delete, 48
calls devaluate, 48
calls exit, 48
calls getTraceOptions, 49
calls getTraceOption, 48
calls hasOption, 49
calls isFunctor, 48
calls lassoc, 48
calls pcounters, 48
calls poundsign, 48
calls ptimers, 48
calls qcar, 48
calls qcdr, 48
calls qslessp, 48
calls resetCounters, 48
calls resetSpacers, 48
calls resetTimers, 48
calls saveMapSig, 49
calls sayKeyedMsg, 48
calls say, 48
calls selectOptionLC, 48
calls seq, 48
calls throwKeyedMsg, 48
calls trace1, 48
calls transTraceItem, 48
calls unabbrev, 48
calls untraceDomainLocalOps, 48
calls untrace, 48
calls vecp, 48
uses $lastUntraced, 49
uses $optionAlist, 49
uses $options, 49
uses $traceNoisely, 49
defun, 48
traceDomainConstructor, 71
calls concat, 71
calls embed, 71
calls exit, 71
calls getOption, 71
calls loadFunctor, 71
calls mkq, 71
calls seq, 71
calls spadTrace, 71
calls traceDomainLocalOps, 71
uses $ConstructorCache, 71
defun, 71
traceDomainLocalOps, 71
calledby traceDomainConstructor, 71
calls sayMSG, 71
defun, 71
tracelet, 85
calls bpiname, 85
calls compileBoot, 85
calls delete, 85
calls gensymp, 85
calls isgenvar, 85
calls lassoc, 85
calls setletprintflag, 85
calls stupidIsSpadFunction, 85
calls union, 85
uses $QuickLet, 85
uses $letAssoc, 85
uses $traceletFunctions, 85
uses $traceletflag, 85
defun, 85
traceOptionError, 59
calls commandAmbiguityError, 59
calls stackTraceOptionError, 59
defun, 59
traceReply, 80
calledby traceSpad2Cmd, 47
calls abbreviate, 81
calls addTraceItem, 81
calls concat, 81
calls exit, 81
calls flowSegmentedMsg, 81
calls isDomainOrPackage, 81
calls isFunctor, 81
calls isSubForRedundantMapName, 81
calls isgenvar, 81
calls poundsign, 81
calls prefix2String, 81
calls qcar, 81
calls rassocSub, 81
calls sayBrightlyLength, 81
calls sayBrightly, 81
calls sayMSG, 81
calls sayMessage, 80
calls seq, 81
calls userError, 81
uses /tracenames, 81
uses $constructors, 81
uses $domains, 81
uses $linelength, 81
uses $packages, 81
defun, 80
traceSpad2Cmd, 47
calledby trace, 47
calls augmentTraceNames, 47
calls getMapSubNames, 47
calls qcar, 47
calls qcdr, 47
calls trace1, 47
calls traceReply, 47
uses $mapSubNameAlist, 47
defun, 47
trademark, 787, 1044
defun, 787
manpage, 1044
transferPropsToNode
called by coerceInt1, 676
called by getSubDomainPredicate, 701
translateTrueFalse2YesNo, 899
called by displaySetOptionInformation, 896
called by displaySetVariableSettings, 897
defun, 899
translateYesNo2TrueFalse, 899
called by initializeSetVariables, 894
called by set1, 1015
called by transOnlyOption, 1053
called by transTraceItem, 60
called by untrace, 59
calls constructor?, 60
calls devaluate, 60
calls domainToGenvar, 60
calls get, 60
calls member, 60
calls objMode, 60
calls objVal, 60
calls throwKeyedMsg, 60
calls transTraceItem, 60
calls unabbrev, 60
calls vcep, 60
uses $doNotAddEmptyModeIfTrue, 60
defun, 60
trapNumericErrors, 1242
defmacro, 1242
tree
called by displaySetVariableSettings, 897
called by initializeSetVariables, 894
called by set1, 1015
trimString
called by checkFilter, 1493
called by removeUndoLines, 41
truth-to-bit, 1231
defmacro, 1231
typeCheckInputAreas, 1443
defun, 1443
types
called by clearCmdParts, 773
unabbrev
called by kDomainName, 1528
called by reportOperations, 1021
called by setExposeAddConstr, 92
called by setExposeDropConstr, 96
called by trace1, 48
called by transTraceItem, 60
unabbrevAndLoad
called by domainToGenvar, 1054
unAbbreviateKeyword, 741
called by doSystemCommand, 718
called by commandsForUserLevel, 741
called by selectOptionLC, 741
called by selectOption, 741
uses $currentLine, 741
uses $syscommands, 741
uses $systemCommands, 741
uses line, 741
defun, 741
underbar
used by writeInputLines, 829
underDomainOf
called by coerceCommuteTest, 690
called by coerceIntByMap, 693
called by coerceIntCommute, 689
called by coerceIntX, 700
called by hasCateSpecialNew, 666
called by retractUnderDomain, 709
INDEX

undo, 36, 1072
calls idenp, 36
calls pname, 36
calls qcar, 36
calls qcdr, 36
calls read, 36
calls stringPrefix?, 36
calls undoSteps, 36
calls userError, 36
uses $InteractiveFrame, 36
uses $options, 36
defun, 36
manpage, 1072
undoChanges, 835
called by undoChanges, 835
called by undoInCore, 834
calls boot-equal, 835
calls putHist, 835
calls undoChanges, 835
uses $HistList, 835
uses $InteractiveFrame, 835
defun, 835
undoCount, 45
called by removeUndoLines, 41
called by undo, 36
calls concat, 45
calls userError, 45
uses $IOindex, 45
defun, 45
undoFromFile, 836
calls assq, 836
calls disableHist, 836
calls exit, 836
calls putHist, 836
calls readHiFi, 836
calls recordNewValue, 836
calls recordOldValue, 836
calls seq, 836
calls updateHist, 836
uses $HiFiAccess, 836
uses $InteractiveFrame, 836
defun, 836
undoInCore, 834
calls assq, 834
calls disableHist, 834
calls putHist, 834
calls readHiFi, 834
calls sayKeyedMsg, 834
calls undoChanges, 834
calls updateHist, 834
uses $HiFiAccess, 834
uses $HistListLen, 834
uses $HistList, 834
uses $IOindex, 834
uses $InteractiveFrame, 834
defun, 834
undoLocalModemapHack, 40
called by undoSingleStep, 39
calls exit, 40
calls seq, 40
defun, 40
undoSingleStep, 38
called by undoSteps, 38
calls assq, 39
calls exit, 39
calls lassoc, 39
calls seq, 39
calls undoLocalModemapHack, 39
defun, 38
undoSteps, 37
called by undo, 36
calls copy, 38
calls qcar, 38
calls qcdr, 38
calls recordFrame, 38
calls undoSingleStep, 38
calls writeInputLines, 37
uses $IOindex, 38
uses $InteractiveFrame, 38
uses $frameRecord, 38
defun, 37
unembed
called by untraceDomainConstructor, 73
unescapeStringsInForm, 287
called by serverReadLine, 269
called by unescapeStringsInForm, 287
calls unescapeStringsInForm, 287
uses $funnyBacks, 287
uses $funnyQuote, 287
defun, 287
unifyStruct, 658
INDEX

called by hasAtt, 654
called by hasCaty, 651
called by hasSig, 653
called by unifyStructVar, 659
called by unifyStruct, 658
calls isPatternVar, 658
calls unifyStructVar, 658
calls unifyStruct, 658
defun, 658

unifyStructVar, 659
called by unifyStruct, 658
calls augmentSub, 659
calls canCoerce, 659
calls constructor?, 659
calls contained, 659
calls containsVars, 659
calls isPatternVar, 659
calls lassoc, 659
calls resolveTT, 659
calls subCopy, 659
calls unifyStruct, 659
local def $hope, 659
local ref $Coerce, 659
local ref $Subst, 659
local ref $domPvar, 659
defun, 659

union
called by breaklet, 86
called by dbShowCons1, 1547
called by getMapSubNames, 65
called by tracelet, 85

unionq
called by getMapSubNames, 65
unparseInputForm, 1238
local def $InteractiveMode, 1238
local def $formatSigAsTex, 1238
defun, 1238

untrace, 59
called by trace1, 48
calls /untrace, 0, 59
calls copy, 59
calls lassocSub, 59
calls removeTracedMapSigs, 60
calls transTraceItem, 59
uses /tracenames, 60
uses $lastUntraced, 60

uses $mapSubNameAlist, 60
defun, 59

untraceDomainConstructor, 73
calls concat, 73
calls delete, 73
calls exit, 73
calls seq, 73
calls unembed, 73
calls untraceDomainConstructor, keepTraced?, 73
uses /tracenames, 73
defun, 73

untraceDomainConstructor, keepTraced?, 73
called by untraceDomainConstructor, 73
calls /untrace, 0, 73
calls boot-equal, 73
calls devaluate, 73
calls exit, 73
calls isDomainOrPackage, 73
calls qcar, 73
calls seq, 73
defun, 73

untraceDomainLocalOps, 71
called by trace1, 48
calls sayMSG, 71
defun, 71

untraceMapSubNames, 1059
called by clearCmdAll, 771
called by clearCmdParts, 773
calls /untrace, 2, 1059
calls assocright, 1059
calls getPreviousMapSubNames, 1059
calls setdifference, 1059
uses $lastUntraced, 1059
uses $mapSubNameAlist, 1059
defun, 1059

unwrap
called by catchCoerceFailure, 692
called by coerceBranch2Union, 698
called by coerceByTable, 691
called by coerceInt0, 674
called by coerceInt1, 676
called by coerceIntX, 700
called by coerceInt, 673
called by retract2Specialization, 704
unwrutable?, 848
INDEX

updateCategoryTable called by loadLib, 1182
called by localrmlib, 1161

calls placep, 848
calls vecp, 848
defun, 848
called by close, 777
called by historySpad2Cmd, 822
called by importFromFrame, 19
called by pathnameConstructorAbbreviations, 756
called by pathnameTypeId, 1191
called by quitSpad2Cmd, 872
called by readSpad2Cmd, 875
called by setOutputAlgebra, 967
called by setOutputFormula, 995
called by setOutputFortran, 974
called by setOutputHtml, 980
called by setOutputMathml, 986
called by setOutputOpenMath, 990
called by setOutputTex, 1002
called by transOnlyOption, 1053
called by yesanswer, 798
calls identp, 1206
defun, 1206

called by addNewInterpreterFrame, 18
called by changeToNamedInterpreterFrame, 17
called by closeInterpreterFrame, 22
called by initializeInterpreterFrameRing, 12

called by nextInterpreterFrame, 17
called by previousInterpreterFrame, 18
called by updateCurrentInterpreterFrame, 16

called by processInteractive, 274
called by undoFromFile, 836
called by undoInCore, 834
called by disableHist, 831
called by startTimingProcess, 831
called by stopTimingProcess, 831
called by updateHistoryHist, 831
called by updateInCoreHist, 831
called by writeHiFi, 831

called by loadLib, 1181
called by localrmlib, 1161
calls clearAllSlams, 1163
calls clearClams, 1163
calls constructor?, 1163

defun, 1163

defun, 1163

defun, 1163

defun, 1163

updateFromCurrentInterpreterFrame, 15
called by addNewInterpreterFrame, 18
called by changeToNamedInterpreterFrame, 17

called by closeInterpreterFrame, 22

called by initializeInterpreterFrameRing, 12

called by nextInterpreterFrame, 17

called by previousInterpreterFrame, 18

called by updateCurrentInterpreterFrame, 16

called by processInteractive, 274

called by undoFromFile, 836

called by undoInCore, 834

called by disableHist, 831

called by startTimingProcess, 831

called by stopTimingProcess, 831

called by updateHistoryHist, 831

called by updateInCoreHist, 831

called by writeHiFi, 831

called by loadLib, 1181

called by localrmlib, 1161

called by clearAllSlams, 1163
called by clearClams, 1163
called by constructor?, 1163

defun, 1163

defun, 1163

defun, 1163

defun, 1163

updateHist, 831

called by loadLib, 1181

called by localrmlib, 1161

called by clearAllSlams, 1163
called by clearClams, 1163
called by constructor?, 1163

defun, 1163

defun, 1163

defun, 1163

defun, 1163

updateDatabase, 1163

called by loadLib, 1181

called by localrmlib, 1161

called by clearAllSlams, 1163
called by clearClams, 1163
called by constructor?, 1163

defun, 1163

defun, 1163

defun, 1163
uses $HistList, 832
defun, 832
updateSourceFiles, 806
called by editSpad2Cmd, 804
called by workfilesSpad2Cmd, 1099
calls insert, 806
calls makeInputFilename, 806
calls member, 806
calls pathnameName, 806
calls pathnameTypeEd, 806
calls pathnameType, 806
calls pathname, 806
uses $sourceFiles, 806
defun, 806
use-fast-links
 called by set1, 1015
userError
called by spadTrace, 67
called by spadUntrace, 78
called by traceReply, 81
called by undoCount, 45
called by undo, 36
userLevelErrorMessage, 722
called by commandUserLevelError, 722
called by optionUserLevelError, 722
calls commandAmbiguityError, 722
calls sayKeyedMsg, 723
calls terminateSystemCommand, 723
uses $UserLevel, 723
defun, 722
util.ht[7.1]
called by buildHtMacroTable, 1333
called by getHtMacroItem, 1334
valid-tokens
 used by token-stack-show, 1117
validateOutputDirectory, 929
called by setFortDir, 930
called by setFortTmpDir, 928
defun, 929
valueArgsEqual?, 695
called by coerceIntByMap, 693
calls algEqual, 696
calls coerceInt, 696
calls getConstructorSignature, 695
calls getdatabase, 695
calls mkObjWrap, 696
calls objValUnwrap, 696
calls replaceSharps, 696
defun, 695
values
called by clearCmdParts, 773
varsInPoly
called by retract2Specialization, 704
vec2list, 1210
called by computeTTTranspositions, 686
defun, 1210
vecp
called by ScanOrPairVec, ScanOrInner, 858
called by basicLookupCheckDefaults, 1188
called by basicLookup, 1186
called by dewritify, dewritifyInner, 854
called by spadClosure?, 853
called by trace1, 48
called by transTraceItem, 60
called by unwritable?, 848
called by writify, writifyInner, 849
vmread, 1228
called by dewritify, dewritifyInner, 854
defun, 1228
voidValue, 1256
defun, 1256
warn
 called by getdatabase, 1156
what, 1089, 1091
calls whatSpad2Cmd, 1091
defun, 1091
manpage, 1089
whatCommands, 1093
called by whatSpad2Cmd, 1091
calls blankList, 1093
calls commandsForUserLevel, 1093
calls filterListOfStrings, 1093
calls sayAsManyPerLineAsPossible, 1093
calls sayKeyedMsg, 1094
calls sayMessage, 1093
calls say, 1093
calls specialChar, 1093
calls strconc, 1093
uses $UserLevel, 1094
uses $linelength, 1094
INDEX

uses $systemCommands, 1094
defun, 1093
whatConstructors, 1097
calledby filterAndFormatConstructors, 1096
calls boot-equal, 1097
calls exit, 1097
calls getdatabase, 1097
calls isort, 1097
calls seq, 1097
defun, 1097
whatSpad2Cmd, 1091
calledby listConstructorAbbreviations, 756
calledby whatSpad2Cmd, 1091
calledby what, 1091
calls apropos, 1091
calls exit, 1091
calls filterAndFormatConstructors, 1091
calls printSynonyms, 1091
calls reportWhatOptions, 1091
calls sayKeyedMsg, 1091
calls selectOptionLC, 1091
calls seq, 1091
calls whatCommands, 1091
calls whatSpad2Cmd, 1091
calls whatSpad2Cmd, fixpat, 1091
calls whatSpad2Cmd, 1091
uses $e, 1091
uses $whatOptions, 1092
defun, 1091
whatSpad2Cmd, fixpat, 1091
calledby whatSpad2Cmd, 1091
calls downcase, 1091
calls qcar, 1091
defun, 1091
whichCat, 587
calledby setMsgForcedAttrList, 586
calledby setMsgUnforcedAttrList, 588
calls ListMember?, 587
uses $attrCats, 587
defun, 587
while, 629, 1190
defmacro, 1190
syntax, 629
whileWithResult, 1190
defmacro, 1190
workfiles, 1099
calls workfilesSpad2Cmd, 1099
defun, 1099
manpage, 1099
workfilesSpad2Cmd, 1099
calledby workfiles, 1099
calls delete, 1099
calls makeInputFilename, 1099
calls namestring, 1099
calls pathname, 1099
calls sayBrightly, 1099
calls sayKeyedMsg, 1099
calls say, 1099
calls selectOptionLC, 1099
calls sortby, 1099
calls specialChar, 1099
calls throwKeyedMsg, 1099
calls updateSourceFiles, 1099
uses $linelength, 1099
uses $options, 1099
uses $sourceFiles, 1099
defun, 1099
wrap, 1193
calledby catchCoerceFailure, 692
calledby mkObjWrap, 447
calledby wrap, 1193
calls lotsof, 1193
calls wrap, 1193
defun, 1193
wrapped2Quote
calledby coerceIntByMap, 693
write-browsedb, 1175
calledby make-databases, 1165
calls allConstructors, 1175
uses *print-pretty*, 1175
uses *sourcefiles*', 1175
uses $spadroot, 1175
defun, 1175
write-categorydb, 1176
calledby make-databases, 1165
calls genCategoryTable, 1176
uses *hasCategory-hash*, 1176
uses *print-pretty*, 1176
defun, 1176
write-interpdb, 1173
calledby make-databases, 1165
uses *ancestors-hash*, 1173
uses *print-pretty*, 1173
uses $spadroot, 1173
defun, 1173
write-operationdb, 1177
calledby make-databases, 1165
uses *operation-hash*, 1177
defun, 1177
write-warmdata, 1178
calledby make-databases, 1165
uses $topicHash, 1178
defun, 1178
writeablep
calledby myWritable?, 1231
writeHiFi, 844
calledby updateHist, 831
calls histFileName, 844
calls object2Identifier, 844
calls rdefostream, 844
calls rshut, 844
calls spadrwrite, 844
uses $HistRecord, 844
uses $IOindex, 844
uses $currentLine, 844
uses $internalHistoryTable, 844
defun, 844
writeHistModesAndValues, 845
calledby processInteractive, 274
calls get, 845
calls putHist, 845
uses $InteractiveFrame, 845
defun, 845
writeInputLines, 828
calledby historySpad2Cmd, 822
calledby saveHistory, 837
calledby undoSteps, 37
calls concat, 828
calls defostream, 828
calls histFileErase, 828
calls histInputFileName, 828
calls namestring, 828
calls readHiFi, 828
calls sayKeyedMsg, 828
calls shut, 828
calls size, 828
calls substring, 828
calls throwKeyedMsg, 828
uses $HiFiAccess, 829
uses $IOindex, 829
uses $NonNullStream, 829
uses $NullStream, 829
defun, 829
writeLib1
calledby saveDependentsHashTable, 1168
calledby saveUsersHashTable, 1168
writeify, 852
calledby safeWritify, 849
calls ScanOrPairVec, 852
calls function, 852
calls writify, writifyInner, 852
uses $seen, 852
uses $writifyComplained, 852
defun, 852
writify, writifyInner, 849
calledby writify, writifyInner, 849
calledby writify, 852
calls boot-equal, 849
calls constructor?, 849
calls devaluate, 849
calls exit, 849
calls hashtable-class, 849
calls hget, 849
calls hkeys, 849
calls hput, 849
calls isDomainOrPackage, 849
calls mkEvalable, 849
calls placep, 849
calls qcar, 849
calls qcdr, 849
calls qrplaca, 849
calls qrplacd, 849
calls qvmaxindex, 849
calls qvmaxindex, 849
calls qvmaxindex, 849
calls qvmaxindex, 849
calls sequ, 849
calls vecp, 849
calls writify, writifyInner, 849
uses $NonNullStream, 850
uses $NullStream, 850
uses $seen, 850
defun, 849
throws, 849
writifyComplain, 848
calls sayKeyedMsg, 848
uses $writifyComplained, 848
defun, 848
xlCannotRead, 316
called by include1, 306
calls inclmsgCannotRead, 316
calls xIMsg, 316
defun, 316
xlCmdBug, 323
called by include1, 307
calls inclmsgCmdBug, 323
calls xIMsg, 323
defun, 323
xlConActive, 318
called by include1, 306
calls inclmsgConActive, 318
calls xIMsg, 318
defun, 318
xlConsole, 319
called by include1, 307
calls inclmsgConsole, 319
calls xIMsg, 319
defun, 319
xlConStill, 319
called by include1, 306
calls inclmsgConStill, 319
calls xIMsg, 319
defun, 319
xlFileCycle, 317
called by include1, 306
calls inclmsgFileCycle, 317
calls xIMsg, 317
defun, 317
xlIfBug, 323
called by include1, 307
calls inclmsgIfBug, 323
calls xIMsg, 323
defun, 323
xlIfSyntax, 322
called by include1, 307
calls Else?, 322
calls Top?, 322
calls inclmsgIfSyntax, 322
calls xIMsg, 322
defun, 322
xlMsg, 311
called by xlCannotRead, 316
called by xlCmdBug, 323
called by xlConActive, 318
called by xlConStill, 319
called by xlConsole, 319
called by xlFileCycle, 317
called by xlIfBug, 323
called by xlIfSyntax, 322
called by xlNoSuchFile, 315
called by xlPrematureEOF, 310
called by xlPrematureFin, 321
called by xlSay, 314
called by xISkippingFin, 320
calls incLine, 311
defun, 311
xlNoSuchFile, 315
called by include1, 306
calls inclmsgNoSuchFile, 315
calls xIMsg, 315
defun, 315
xlOK, 311
called by include1, 306
calls xIK1, 311
defun, 311
xlOK1, 311
called by include1, 306
called by xlOK, 311
calls incLine1, 311
defun, 311
xlPrematureEOF, 310
called by include1, 306
calls inclmsgPrematureEOF, 310
calls xIMsg, 310
defun, 310
xlPrematureFin, 321
called by include1, 307
calls inclmsgPrematureFin, 321
calls xIMsg, 321
defun, 321
xlSay, 314
called by include1, 306
calls inclmsgSay, 315
calls xIMsg, 314
defun, 314
xISkip, 314
called by include1, 306
  calls concat, 314
  calls incLine, 314
  defun, 314

xlSkippingFin, 320
  called by include1, 307
  calls inclmsgFinSkipped, 320
  calls xMsg, 320
  defun, 320

xTokenreader, 1114
  used by init-boot/spad-reader, 1114
  defvar, 1114

yesanswer, 798
  called by displayOperations, 798
  calls queryUserKeyedMsg, 798
  calls string2id-n, 798
  calls upcase, 798
  defun, 798

ySearch, 1500
  called by conPage, 1503
  calls constructorSearch, 1500
  defun, 1500

zeroOneConvert
  called by kePageOpAlist, 1511

zeroOneTran, 292
  called by intInterpretPform, 292
  calls nsubst, 292
  defun, 292